Deep learning for human engineered systems: Weak supervision, interpretability and knowledge embedding

Sambuddha Ghosal

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Computer Sciences Commons, Materials Science and Engineering Commons, Mechanical Engineering Commons, and the Mechanics of Materials Commons

Recommended Citation


This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Deep learning for human engineered systems: Weak supervision, interpretability and knowledge embedding

by

Sambuddha Ghosal

A dissertation submitted to the graduate faculty in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Mechanical Engineering

Program of Study Committee:
Soumik Sarkar, Major Professor
Baskar Ganapathysubramanian
Arti Singh
James Michael
Jin Tian

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2019

Copyright © Sambuddha Ghosal, 2019. All rights reserved.
DEDICATION

....Dedicated to my parents: Alok Ghoshal and Sumana Ghosal

My aunt, Dr. Kalpana Ghoshal

&

My soulmate and better-half, Miss Olivia Biswas
# TABLE OF CONTENTS

| LIST OF TABLES | vi |
| LIST OF FIGURES | vii |
| ACKNOWLEDGEMENTS | xvi |
| ABSTRACT | xvii |
| **CHAPTER 1. INTRODUCTION** | 1 |
| 1.1 Deep Learning for Human engineered systems | 5 |
| 1.1.1 Generating an automated labelling framework for efficient detection of crops and providing accurate yield estimate | 6 |
| 1.1.2 Deep Learning for exploring structure-property maps in photovoltaics | 7 |
| 1.1.3 Top-K Feature Maps: Semi-supervised framework for explainable Deep Learning | 8 |
| 1.1.4 Knowledge Embedding using Deep Learning | 9 |
| 1.2 Physics augmented Deep Learning | 10 |
| **CHAPTER 2. PRELIMINARIES ON DEEP LEARNING AND TRAINING CONSIDERATIONS** | 11 |
| 2.1 Restricted Boltzmann Machines (RBMs) | 11 |
| 2.2 Deep Neural Networks | 13 |
| 2.3 Deep Convolutional Neural Networks (DCNNs) | 14 |
| 2.3.1 Overview of the different layers used in DCNNs | 17 |
| 2.4 Convolutional Autoencoders | 19 |
| 2.5 Training Considerations | 20 |
| 2.5.1 Data Preprocessing | 20 |
| 2.5.2 Batch Normalization | 21 |
| 2.5.3 Regularization: Dropout | 22 |
| 2.5.4 Activation Functions | 22 |
| 2.6 Convolutional Encoder-Decoders in solving PDEs: | 24 |
| **CHAPTER 3. A WEAKLY SUPERVISED DEEP LEARNING FRAMEWORK FOR SORGHUM HEAD DETECTION AND COUNTING** | 27 |
| 3.1 Introduction | 27 |
| 3.2 Field experiment, Data Collection and Preparation | 31 |
| 3.3 Methods | 32 |
| 3.3.1 Deep Learning Framework | 32 |
| 3.3.2 Reducing human labeling cost via a weak supervision strategy | 37 |
5.3.3 DCNN explanation framework for severity classification and quantification . . 95
5.4 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
5.4.1 Stress Identification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
5.4.2 Symptom Explanation and Severity Quantification . . . . . . . . . . . . 102
5.4.3 High-throughput Deployment and Transfer Learning Capability . . . . . . . 103
5.5 Discussions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103
5.5.1 Severity Rating by Human Experts using a Visual Application . . . . . . . 106
5.5.2 Failure Cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107

CHAPTER 6. ENCODING CNN-SURROGATES AS INVARIANCE-CHECKERS IN DEEP
GENERATIVE MODELS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119
6.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119
6.1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119
6.1.2 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120
6.2 Backbone to S-InvNet: The InvNet Algorithm . . . . . . . . . . . . . . . . . . . . . . 121
6.3 S-InvNet: Generating Microstructures using CNN-surrogate as Invariance to InvNet 123
6.4 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125
6.5 S-InvNet II . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 128

CHAPTER 7. INCORPORATION OF DOMAIN KNOWLEDGE IN DEEP LEARNING
FRAMEWORKS: PHYSICS CONSTRAINED DEEP LEARNING FOR SURROGATE
MODELLING . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 130
7.1 Network Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 131
7.2 Problem Definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132
7.2.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132
7.2.2 Generating Solutions for PDEs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 133
7.2.3 Data-set Generation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 135
7.3 Evaluation Metric: Relative $\ell_2$ Error . . . . . . . . . . . . . . . . . . . . . . 136
7.4 Results for ED framework optimized with with Combined Loss, $L$ . . . . . . 136
7.4.1 Results for Viscid Flow, $\lambda = 1$, Combined Loss function . . . . . . . 137
7.4.2 Results for Viscid Flow, $\lambda = 10$, Combined Loss Function . . . . . . 149
7.4.3 Results for Inviscid Flow, $\lambda = 1$, Combined Loss Function . . . . . . 159
7.4.4 Results for Inviscid Flow, $\lambda = 10$, Combined Loss Function . . . . . 162
7.5 Comparison between Results for ED framework optimized with with Alternating and
Summed Loss terms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 165

CHAPTER 8. CONCLUSION AND FUTURE DIRECTIONS . . . . . . . . . . . . . . . . . . . . 168
8.1 Future Directions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 168

REFERENCES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 172
LIST OF TABLES

Table 4.1 Effect of depth of network on generalizability . . . . . . . . . . . . . . . . . 84
LIST OF FIGURES

Figure 1.1 An Illustration of a Deep Learning (DL) Tool Chain from Data Gathering to Decision Making. A key first step is to gather a diverse set of data, followed by training of the DL model. The trained model is then finally tested on the unseen test image data subset. After successful completion of this testing, the trained DL model can be deployed for inference. Training a DL model can be resource intensive but on the other hand, deploying a trained model is relatively simple. ................................................................. 2

Figure 2.1 Differentiation between Boltzmann machines and Restricted Boltzmann machines (RBMs). In RBMs, there are no interconnections between the nodes in the same layer i.e., there are no visible-visible or hidden-hidden connections. ................................................................. 12

Figure 2.2 Stacking of RBMs to form a Deep Neural Network ........................................... 13

Figure 2.3 Illustration showing how 2D Convolution works ............................................. 15

Figure 2.4 Illustration showing how 2D Max-Pooling works ............................................ 16

Figure 2.5 Illustration of a typical Convolutional Neural Network architecture ................ 17

Figure 2.6 Illustration of a typical Convolutional Auto-Encoder architecture ................. 20

Figure 2.7 Data Preprocessing techniques. Illustration Courtesy: Stanford CS231n lecture notes ................................................................. 21

Figure 2.8 The idea behind performing Batch Normalization. Illustration Courtesy: Stanford CS231n lecture notes ................................................................. 22

Figure 2.9 Dropout takes care of over-fitting issues by randomly dropping activation units. Illustration Courtesy: Stanford CS231n lecture notes ................................................................. 23

Figure 2.10 (a) A dense block contains L = 3 layers h1, h2, h3 with growth rate K = 2. (b) The second layer h2 of the dense block, where x2 = h2([x1, x0]) is its output feature map. Notice that the input to the third layer is the concatenation of the output and input features of h2, i.e. [x2, x1, x0]. As is often the case, each layer is composed of Batch Normalization, Rectified Linear Unit (ReLU) and Convolution (Conv). The convolution kernel has kernel size k = 3, stride s = 1 and zero padding p = 1, which keep the size of the feature maps the same as the input. Source and further details: (Zhu and Zabaras (2018) ................................................................. 25
Figure 2.11 Both (a) encoding layer and (b) decoding layer contain two convolutions. The first convolution reduces the number of feature maps while keeping their size the same using a kernel with parameters $k = 1, s = 1, p = 0$; the second convolution changes the size of the feature maps but not their number using a kernel $k = 3, s = 2, p = 1$. The main difference between (a) and (b) is in the type of the second convolution layer, which is Conv and ConvT respectively, for downsampling and upsampling. Note that no pooling is used at transition layers for maintaining the location information. The colored feature maps used here are independent from the feature maps with the same color shown in other figures. Source and further details: Zhu and Zabaras (2018).

Figure 3.1 Web-based annotation tool used for labeling data. Here, the left panel shows labeling site, where labeling is done from scratch and the right panel shows validation site, where we validate the pre-labeled data (pre-labeled either by human annotators or the deep learning model).

Figure 3.2 RetinaNet: The deep learning architecture adopted in our framework.

Figure 3.3 Automated annotation protocol: The initial input image passed through the network is part of the training data-set. This trains the RetinaNet architecture and generates a model, which then is used to annotate new data. This annotation is then verified by human expert raters and fed back into the network as new training data. Process is repeated until network reaches desired level of accuracy.

Figure 3.4 Variation of $R^2$ for the 1260-test data set with change in training data size (following our automated annotation protocol (see Fig. 3.3), varying from models trained with 1, 2, 3, 4, 5, 10, 20, 30, 40, 50 and 283 images.

Figure 3.5 (A) Model output image (about 180 seconds for verifying and making corrections) vs (B) Manually annotated image starting from scratch (about 720 seconds to label the entire image).

Figure 3.6 Comparing results for a randomly selected test sample (genotype G357) for different models - starting from model trained with one sample and passed through our weak supervision based learning protocol using 2, 3, 4, 5, 10, 20, 30, 40, 50 and 283 training samples.

Figure 3.7 Correlation plots for: Test 1260-image test data-set (left) with $\rho = 0.9389$, $R^2 = 0.8815$ @ mAP of 0.9413 (IoU threshold = 0.5), test data-set labeled “52Cropped” (right-top) with $\rho = 0.9063$, $R^2 = 0.8214$ and test data-set labeled “40Plots” (right-bottom) with $\rho = 0.8756$, $R^2 = 0.7667$.

Figure 3.8 Model output for a sample from the “52Cropped” data.

Figure 3.9 Feature Map Visualization for test image from genotype G40.

Figure 3.10 Feature Map Visualization for brown sorghum heads (top) and white sorghum heads (bottom).
Figure 3.11 Model output for varying orientations. Here we take the original image from the test set and perform several counterclockwise (CCW) rotations and then test our model on these images. As can be seen from the detection results, the model accurately picks up on all the sorghum heads in the original as well as the transformed images.

Figure 3.12 More detection results for Sorghum Plot images across different genotypes and variations.

Figure 3.13 Variation of $R^2$ for the 1260-test data set with change in orientation. The $R^2$ values are for the trained model (trained with 40 images).

Figure 3.14 The variation of mAP with respect to IoU shows that mAP begins to drop drastically after the 0.6 IoU threshold. Thus, for simplicity and getting satisfactory outputs from the model, an IoU threshold of 0.5 is chosen, which is a widely accepted threshold in the deep learning and object detection/localization community.

Figure 3.15 Unseen Case: Genotype G286 (white-heads). Although the model had no white-head samples to train on, it is still able to detect some as seen in the sample to the right.

Figure 3.16 Feature Map Visualization for brown sorghum head (left) and white sorghum head (right). For the brown sorghum head, we see that framework specifically picks up the region around the heads while it does not do the same for the white heads.

Figure 4.1 DLSP (Deep Learning for Structure Property) framework: We construct a forward map from morphology to performance. Upon building trust in this trained model, we use it for performing manual exploration and insight buildings, as well as and automated design.

Figure 4.3 Proposed architecture and in-sample testing results.

Figure 4.4 Saliency maps and performance of our custom trained CNN. Note how the saliency maps closely follow the interface regions in the microstructure. It should also be noted that the networks shows good performance even on samples outside the training data set.

Figure 4.5 Comparison of Saliency map outputs for our Custom Model (second column), VGG-net (third column) and ResNet-50 (fourth column), with input morphologies shown in the first column: top row shows an example image for class 0, bottom row shows an example image from fractal-like morphologies (correctly predicted as class 9 by our custom model).

Figure 4.7 Manual exploration and insight building using the browser interface. Notice how several physics based intuitive trends can be identified and understood by incrementally perturbing the original bilayer morphology.

Figure 4.8 Exploration by semi-automated design: The optimization started with a bilayer structure. Notice how the framework directs the formation of finer features. Figure (d) shows the simulated electron and hold current densities under short circuit conditions for this optimized morphology. The result from automated design has been modified using physics based principles.
Figure 4.10 Both the standard architectures show performance similar to our custom architecture. But these do not provide any meaningful explanations to their predictions (Fig. 4.4)

Figure 5.1 Schematic illustration of foliar plant stresses in soybean grouped into two major categories, biotic (bacterial and fungal) and abiotic (nutrient deficiency and chemical injury) stress. The images were used to develop the DCNN for the following eight stresses: bacterial blight (Pseudomonas savastanoi pv. glycinea), bacterial pustule (Xanthomonas axonopodis pv. glycines), sudden death syndrome (SDS, Fusarium virguliforme), Septoria brown spot (Septoria glycines), frogeye leaf spot (Cercospora sojina), IDC, potassium deficiency and herbicide injury. For each stress, information such as symptom descriptors, areas of appearance and most commonly mistaken stresses that exhibit similar symptoms are listed. These particular foliar stresses were chosen because of their prevalence and confounding symptoms.

Figure 5.2 Leaf Imaging Platform

Figure 5.3 Example leaf images of a healthy leaf and each stress class

Figure 5.4 Illustration of data augmentation scheme

Figure 5.5 (a) DCNN architecture used. (b) Explanation Phase. The concept of isolating the top-K high-resolution feature maps learnt by the model based on their localized activation levels was applied to automatically visualize important image features used by the DCNN model.

Figure 5.6 Probability distribution of healthy leaf activations

Figure 5.7 Illustration of the concept of isolating top-K high-resolution feature maps learnt by the model based on their localized activation levels

Figure 5.8 Examples of feature map ranking based on activation energy content

Figure 5.9 Leaf image examples for each soybean stress correctly identified by the DCNN model. The unsupervised explanation framework is applied to isolate the regions of interest (symptoms) extracted by the DCNN model, that are highly correlated (spatially) with the symptoms marked manually by expert raters.

Figure 5.10 This confusion matrix shows the stress classification results of the DCNN model for eight different stresses and healthy leaves. The overall classification accuracy of the model is 94.13%. The greatest confusion among stresses was found among bacterial blight, bacterial pustule, and Septoria brown spot which can be attributed to the similarities in symptom expression among these stresses.
Figure 5.11 Distributions of spatial correlation between human marked symptoms and machine learning explanations for four stresses: Septoria brown spot, IDC, Herbicide injury and Sudden Death Syndrome. As the distributions are significantly skewed towards high correlation values, they show the success of the DCNN-based severity estimation framework to correctly identify symptoms for these stresses. Few examples with different severity classes (machine learning explanations on left, actual images on right) using a standard discretized severity scale (0-25%: resistant, 25-50%: moderately resistant, 50-75%: susceptible, and 75-100%: highly susceptible).

Figure 5.12 Examples of xPlNet results: Rated leaf images for four different soybean stresses.

Figure 5.13 Variation of test accuracy w.r.t. the increasing size of training data.

Figure 5.14 Visualizing learnt filters for the input image shown here (Conv. represents a 2D convolutional layer, and Pool. represents a 2D max-pooling layer).

Figure 5.15 xPlNet performance on non-destructive leaf image samples for IDC, Potassium deficiency and SDS; top row shows the original images either with clipboard imaging protocol (IDC) or canopy imaging protocol (Potassium deficiency and SDS); middle row shows the extracted leaf images used for testing; bottom row shows the xPlNet results.

Figure 5.16 This table illustrates the ability of the framework to accurately identify similar stress symptoms used in training the DCNN model, but in non-soybean crops, such as IDC in cucurbits and potassium deficiency in oilseed rape.

Figure 5.17 Design of the web app for expert-based foliar stress identification and severity estimation.

Figure 5.18 Illustration showing the tagging process.

Figure 5.19 Scatter plot comparing the severity ratings of the same images between two raters for four stresses (IDC, SDS, potassium deficiency and Septoria brown spot) that were pooled and solid red line is the 45° line. The results indicated high inter-rater variability between experienced raters, especially as the stress severity of leaf images increase.

Figure 5.20 Examples in which xPlNet failed to detect proper stress signatures.

Figure 6.1 S-InvNet: The proposed architecture for online targeted microstructure generation. The CNN surrogate (working in tandem with the Discriminator) is used as the invariance checker which endures that the Generator output belongs to the target class.

Figure 6.2 S-InvNet II: We modify the previous S-InvNet architecture by utilizing the trained surrogate as a filter to the true microstructure data, before it is passed to the Discriminator. The surrogate allows only the target data (i.e., the data belonging to the target class) to be passed to $D(.)$.

Figure 6.4 Samples from Surrogate-InvNet (S-InvNet) generated microstructures for target class 7 and corresponding predictions (population size: 64 samples).
Figure 6.6  Samples from Surrogate-InvNet (S-InvNet) generated microstructures for target class 8 and corresponding predictions (population size: 64 samples). 126

Figure 6.8  Samples from Surrogate-InvNet (S-InvNet) generated microstructures for target class 9 with corresponding predictions (population size: 64 samples). 127

Figure 6.10 Samples from Surrogate-InvNet (S-InvNet) generated microstructures for target class 9 with variations in presence of training data classes (population size: 64 samples). 127

Figure 6.12 Samples from Surrogate-InvNet II (S-InvNet II) generated microstructures for target class 9 with variations in presence of training data classes (population size: 64 samples). 128

Figure 7.1  Network architecture: DenseED. Figure design idea sourced from Zhu and Zabaras (2018). 131

Figure 7.2  Dense convolutional encoder-decoder network as the deterministic surrogate. The models input is the realization of a random field, the models output is the prediction for each input field. The model is trained with physics-constrained loss without target data. Figure design idea sourced from Zhu et al. (2019). 132

Figure 7.3  Inputs to the Model - 2 different cases. 133

Figure 7.4  Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row) leftmost column shows the Input Field (the input to the ED framework), middle column shows the ED network solution and the rightmost column shows the relative $\ell_2$ error between the Exact numerical solution and the ED output. Parameters: $[\lambda = 1, \nu = 0.01, z = 2]$. 138

Figure 7.5  Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.01, z = 4]$. 139

Figure 7.6  Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.01, z = 5.5]$. 140

Figure 7.7  Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.01, z = 6]$. 140

Figure 7.8  Temporal variation for the predicted solution for Input Field 1 with parameters $\lambda = 1, \nu = 0.1$ - time axis corresponds to a range of $t = 0 - 0.2s$. 141

Figure 7.9  Temporal variation for the predicted solution for Input Field 2 with parameters $\lambda = 1, \nu = 0.1$ - time axis corresponds to a range of $t = 0 - 0.2s$. 141

Figure 7.10 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0225, z = 2]$. 142

Figure 7.11 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0225, z = 4]$. 142
Figure 7.12 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0225, z = 2]$.

Figure 7.13 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0225, z = 6]$.

Figure 7.14 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0725, z = 2]$.

Figure 7.15 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0725, z = 4]$.

Figure 7.16 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0725, z = 5.5]$.

Figure 7.17 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0725, z = 6]$.

Figure 7.18 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.1, z = 2]$.

Figure 7.19 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.1, z = 4]$.

Figure 7.20 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.1, z = 5.5]$.

Figure 7.21 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.1, z = 6]$.

Figure 7.22 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.01, z = 2]$.

Figure 7.23 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.01, z = 4]$.

Figure 7.24 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.01, z = 5.5]$.

Figure 7.25 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.01, z = 6]$.
Figure 7.26 Temporal variation for the predicted solution for Input Field 1 with parameters $\lambda = 10, \nu = 0.1$ - time axis corresponds to a range of $t = 0 - 0.2s$ ... 151

Figure 7.27 Temporal variation for the predicted solution for Input Field 2 with parameters $\lambda = 10, \nu = 0.1$ - time axis corresponds to a range of $t = 0 - 0.2s$ ... 152

Figure 7.28 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0225, z = 2]$ ... 152

Figure 7.29 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0225, z = 4]$ ... 153

Figure 7.30 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0225, z = 5.5]$ ... 153

Figure 7.31 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0725, z = 6]$ ... 154

Figure 7.32 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0725, z = 2]$ ... 154

Figure 7.33 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0725, z = 4]$ ... 155

Figure 7.34 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0725, z = 5.5]$ ... 155

Figure 7.35 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0725, z = 6]$ ... 156

Figure 7.36 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.1, z = 2]$ ... 156

Figure 7.37 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.1, z = 4]$ ... 157

Figure 7.38 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.1, z = 5.5]$ ... 157

Figure 7.39 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.1, z = 6]$ ... 158
Figure 7.40 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0, z = 2]$.

Figure 7.41 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0, z = 4]$.

Figure 7.42 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0, z = 5.5]$.

Figure 7.43 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0, z = 6]$.

Figure 7.44 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0, z = 2]$.

Figure 7.45 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0, z = 4]$.

Figure 7.46 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0, z = 5.5]$.

Figure 7.47 Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0, z = 6]$.

Figure 7.48 A comparison between absolute errors between ED output and exact numerical solution for combined loss and alternating loss optimization for Input Field 1. The values beneath each sub-figure represent the relative $\ell_2$ error between the network output and the corresponding exact solution.

Figure 7.49 A comparison between absolute errors between ED output and exact numerical solution for combined loss and alternating loss optimization for Input Field 2. The values beneath each sub-figure represent the relative $\ell_2$ error between the network output and the corresponding exact solution.
ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research throughout my graduate education and the writing of this thesis.

First and foremost, Dr. Soumik Sarkar for his impeccable guidance, patience and support throughout my graduate research carrier, providing me assistantship and the writing of this thesis. His insights, experience and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. Furthermore, it is through him that I have come to know about the intricacies of pursuing a career in independent as well as collaborative research and academia further down the line.

I would also like to thank my committee members, Dr. Baskar Ganapathysubramanian, Dr. Arti Singh, Dr. James Michael and Dr. Jin Tian for their valuable efforts and contributions in helping me to bring this work to fruition. Dr. Ganapathysubramanian and Dr. Singh have been pillars of support in building my early stages of my graduate life when I was just a freshman and new to the field of research and writing scientific papers and journals. It is through them and with their collaborative support that I was eventually able to publish some of my best works during my time as a graduate student. I would additionally like to thank Dr. James Michael and Dr. Jin Tian for their continual guidance throughout the mid and later stages of preparing my thesis and providing crucial feedback that led to a significant improvement of the quality of this thesis.

Finally, I would like extend my sincerest gratitude to my teammates, collaborators (both within the U.S.A and abroad from India, Japan, China, France and Australia) whom I have worked with closely and my fellow lab members as well as the organizations who helped fund my research throughout the years.
ABSTRACT

Pattern recognition has its origins in engineering while machine learning developed from computer science. Today, artificial intelligence (AI) is a booming field with many practical applications and active research topics that deals with both pattern recognition and machine learning. We now use software and applications to automate routine labor, understand speech (using Natural Language Processing) or images (extracting hierarchical features and patterns for object detection and pattern recognition), make diagnoses in medicine, even intricate surgical procedures and support basic scientific research.

This thesis deals with exploring the application of a specific branch of AI, or a specific tool, Deep Learning (DL) to real world engineering problems which otherwise had been difficult to solve using existing methods till date. Here we focus on different Deep Learning based methods to deal with several such problems. We also explore the inner workings of such models through an explanation stage for each of the applied DL based strategies that gives us a sense of how such typical black box models work, or as we call it, an explanation stage for the DL model.

This explanation framework is an important step as previously, Deep Learning based models were thought to be frameworks which produce good results (classification, object detection, object recognition to name a few), but with no explanations or immediately visible causes as to why it achieves the results it does. This made Deep Learning based models hard to trust amongst the scientific community. In this thesis, we aim to achieve just that by deploying such explanation frameworks, which will be discussed later in the subsequent chapters. We use one such explanation framework to develop a surrogate model to predict properties of microstructures as well.

Furthermore, we dig deep into the realm of semi-supervised or weakly supervised learning which utilizes minimal data for its training phase and develop a novel framework capable of accurately predicting yields of crops like sorghum. We utilize this framework to learn from available data in an
online fashion, annotate new data, significantly reduce manual annotation time and at the same
time predict crop yield, a first of its kind within the target application domain.

We also propose a new generative modeling approach, Surrogate Invariance Network (S-InvNet),
that can efficiently model data spaces with known invariances. We devise an adversarial training
algorithm to encode such invariances into the data distribution. We validate our framework by
reconstructing two-phase microstructures with desired physical properties. The physical properties
are governed by a data-driven surrogate model which acts as the invariance for our case. In essence,
we fuse a generative and a classification model with the classification model acting as an invariance
checker that enforces the property of the target microstructure.

How a Deep Learning model can iteratively solve a non-linear Partial Differential Equation
(PDE) based on a given set of initial conditions and using a Physics-governed loss function (instead
of traditional loss functions used to optimize Deep Learning models) is presently an area of great
research interest. We explore this as well. Previous literature deal with utilizing a single loss
function to solve such examples of PDE-governed systems. Moreover, the most recent work proposes
to solve only a simple spatially varying PDE (Darcy Flow). We extend the framework to deal
with both spatial and time-varying PDEs (Burgers’ Equation). Furthermore, we also propose an
alternating minimization process for optimizing neural networks as PDE-solvers. This alternatively
minimizes two separate loss functions. We note that this process works just as well as the single
loss minimization and in certain cases, performs even better. From a neural network standpoint, we
use a Convolutional Encoder-Decoder framework as our PDE-surrogate.
CHAPTER 1. INTRODUCTION

Machine Learning (ML) is a multidisciplinary approach to data analysis based on probability theory, optimization, decision theory, statistics and visualization. ML approaches are typically useful in situations where large amounts of data are available, relating inputs (for instance, images) to output quantities of interest (classes in case of a classification task). ML enables users to discover patterns and govern discovery by simultaneously looking at a combination of factors instead of analyzing each data feature individually. This was previously a major bottleneck because the high dimensionality of data makes them extremely difficult to analyze through classical techniques. Another key challenge is that the underlying processes for linking the inputs to the outputs are too complex to model mathematically. ML methods have been applied with spectacular results to problems previously presumed to be impossible to model. The success of ML tools is attributed to their ability to identify a hierarchy of features and generalized trends from available data. These tools have also proved particularly adept at integrating disparate and often redundant data to draw coherent (and often non-intuitive) patterns for identification and quantification. Current progress in ML has resulted in scalable, robust, and flexible software tools as well.

Deep Learning (referred to as “DL” from here on) is a class of ML techniques which utilizes a stack of multiple processing layers where the output from the previous layer is used by each succeeding layer as input to learn representations of data with multiple levels of abstraction 1.1. DL models are typically built using multilayer neural networks where two adjacent layers of features are connected by neurons that essentially represent nonlinear transformations. Model parameters that govern the network’s performance include weights and biases of the neurons that are multiplied and added to the input. In the forward direction, the input data goes through a layerwise transformation until the target (decision making) layer is reached. In an image classification problem for example, an input image is transformed using these hierarchical nonlinear transformations and the image
class is obtained as the output at the target layer. Training such a model begins with a set of initial parameters (randomly chosen for most purposes) for the deep neural network (DNN). Errors are computed at the target layer between the actual outputs and the desired outputs given by the training data labels for a large number of examples. Then the errors are used in a feedback mechanism in a layer-by-layer fashion (from target to input) to update the parameters until a satisfactory level of decision accuracy is achieved at the target layer. A method called the error back-propagation algorithm is used for this training process. DL models can be trained in both supervised and unsupervised ways. In supervised DL, labeled input data are mapped to output via a weights vector and errors are back-propagated (adjusting the weights) from the output layer to the input; whereas, in the case of unsupervised DL, the objective is to identify patterns from the data for various purposes, such as clustering. Training DL models typically involve Stochastic Gradient Descent (SGD), an iterative optimization algorithm (or variants thereof, such as ADAM (Kingma and Ba (2014a)), to accomplish the back-propagation-based (to update weights) model parameter learning. However, the choice of hyperparameters in the training process determines (to a large extent) a successful DNN model.

Figure 1.1: An Illustration of a Deep Learning (DL) Tool Chain from Data Gathering to Decision Making. A key first step is to gather a diverse set of data, followed by training of the DL model. The trained model is then finally tested on the unseen test image data subset. After successful completion of this testing, the trained DL model can be deployed for inference. Training a DL model can be resource intensive but on the other hand, deploying a trained model is relatively simple.
DL, in general is a very powerful tool that deals with solving tasks that are easy for people to perform but at the same time hard for people to describe formally i.e., problems that are more intuitive, for instance, recognizing words or faces in images. This kind of methods allow computers to learn from experience and understand their surroundings in terms of hierarchy of concepts, with each concept defined through its relation to simpler concepts. This hierarchy of concepts enables the computer to learn complicated concepts by building them out of simpler ones.

In recent times, DL has proved its mettle in various fields of modern science such as medical imaging applications that have achieved dermatologist level classification accuracies for skin cancer (Esteva et al. (2017)), in modeling neural responses and population in visual cortical areas of the brain (Yamins and DiCarlo (2016)) and in predicting sequence specificities of DNA- and RNA-binding proteins (Alipanahi et al. (2015)). Similarly, deep learning based techniques have made transformative demonstration in the context of performing complex cognitive tasks such as achieving human level or better accuracy for playing Atari games using Deep Q network (Mnih et al. (2015)) and beating a human expert in playing the Chinese game of Go (Silver et al. (2016)). DL models have been shown to outperform other state of the art techniques in handling and analyzing large dimensional data (both spatial and temporal) as well, by learning the hierarchical features to perform various tasks such as, classification and bulk structure detection given a large corpus of data. DL on its own has recently seen a lot of success thanks to the rapid development of hardware that has enabled training of large data sets and also the recent boom in Machine Learning (ML) and DL-theory advances. While applications like playing games are mostly generic in nature and do not involve any safety-critical or time-critical aspects to it, applications like health diagnostics, perception of autonomous vehicles to name a few. As such, the Deep Learning community has shown an increasing interest to build interpretability mechanisms to build user’s trust with the so-called "black-box" DL models.

However, understanding the inner workings of a DL model has proven to be challenging. This has led to many experts in varied fields of study touting applied DL-based models as a "black-box". Thus, when it comes to applying and using DL-based frameworks as end-to-end packages that can
be used for real-world scientific and/or engineering applications, users and experts find it difficult to put their complete trust in such so-called “black-box” models. To circumvent this issue, researchers in the DL community have come up with several explanatory techniques for traditional DL-based methods to give a sense of explainability to these models. Introduction of such explainability provides trust to some extent and many such interpretability mechanisms have been developed to explain behaviors of DL models for instance, deep Convolutional Neural Networks (CNNs) which are extensively used for image/object classification and/or detection tasks. Having said that, such methods are still ad-hoc and lack robustness and more often tend to vary from one application to another. This is because at present, there is still a lack of consensus in the DL community about the specific targets and performance measures that can properly evaluate these interpretability mechanisms.

**Human Engineered Systems** are engineering, technological and scientific marvels that exist today in the modern world. These systems are governed by the physical and system-specific laws that govern them based on existing parameters and constraints that they are bound to. The advent and advancement of data and ML/DL-based methods has recently enabled experts and researchers as well as engineers looking to incorporate and adopt them to existing and upcoming technological and engineering innovations. Such a merger would eventually lead to more robust, tunable, evolving and smart systems that can utilize the intrinsic information embedded in data to make high level decisions. We can make such systems trustworthy as well by providing explanation/interpretability mechanisms. In cases, extensive data is not always available and as such, researchers have to come up with alternative methods that utilize minimum data or in cases, no data at all (random initialization). This leads to crafting of unsupervised learning techniques as well as frameworks that utilize sparse data sets and then incrementally learn and build upon those data sets in an active manner to produce a fully-optimized and robust framework that can then be used for certain engineering purposes.

We first start with the interpretability aspect of this work, where a question arises as to whether there can be a manual verification/evaluation step added to the existing DL framework which
adds to the trust and interpretability factor which otherwise has been severely lacking for such models. A possible solution proposed is to introduce a Human in the Loop (HiL) method while developing DL applications and frameworks. Along with the trust factor, there comes a necessity for Deep Learning models to be evolutionary and self-sustaining. What that means is that the preliminary stages of procuring the data necessary for executing the training step for developing the end-to-end DL model has to be automated as well. We thus propose an extension of our trust mechanism that enables automated labelling of data as we see in Section 1.1.1, where we develop a weakly-supervised framework which is capable of handling raw data and labelling that the new data set which is then fed back into the Human in the Loop Deep Learning (HiLDL) framework, thus completing the feedback loop that eventually trains and validates the network to achieve superior performance. The unique aspect of this kind of modelling is that it is self-sustaining i.e., it only requires an initiation and then as new data comes in, it can automatically take care of itself after a break-even point is decided based on the desired performance metric tailored to its specific application area. This acts as a learning framework where we have a minimum need for human supervision. Another weakly-supervised learning framework that is discussed briefly in Section 1.1.3 and in detail in Chapter 5, is Top-K explanation framework which uses learnt feature maps extracted from a typical DL framework to quantify and disease severity in plants. Then comes knowledge embedding using DL frameworks where, in Chapter 6, we explore how Deep Learning frameworks can be leveraged to solve existing Partial differential equations (PDEs). Further extending the concept of knowledge embedding, in Chapter 6, we encode material properties of microstructures using Deep Learning frameworks and come up with a surrogate model that simply uses existing data and models underlying complex microstructure configurations based on achieving a target microstructure with desirable properties.

1.1 Deep Learning for Human engineered systems

Moving forward, a short introduction to the concept of Human-in-the-Loop (HiL) Deep Learning (HiLDL) is put forth. As an overview, in this chapter, we briefly discuss 4 practical applications
which are further elaborated in the Chapters 3, 4, 5, 6 and 7. We here show how involving Human experts and embedding existing knowledge (based on the problem at hand) for this exact purpose within a typical DL-framework training-validation-testing loop can take care of the issues raised and provides that trust factor as well as acts as a intermediate performance measurement step for such models.

1.1.1 Generating an automated labelling framework for efficient detection of crops and providing accurate yield estimate

This is discussed in detail in Chapter 3, where we choose a real world problem and in the process of solving the problem, discuss how fusing human expertise while training a DL model (RetinaNet, which is an object detection framework) can effectively improve the quality of a Deep Learning model exponentially and can provide for a much better and significantly more robust framework.

The yield of cereal crops such as sorghum (Sorghum bicolor L. Moench) depends on the distribution of crop-heads in varying branching arrangements. Therefore, counting the head number per unit area is critical for plant breeders to correlate with the genotypic variation in a particular breeding field. However, to measure such phenotypic traits manually is an extremely labor-intensive process and suffers from low efficiency and human errors. Moreover, the process is almost infeasible for large-scale breeding plantations or experiments (for instance, cases where there are more than 1000 plots). Thanks to the rapid development of computer vision techniques such as deep convolutional neural network (CNN) based object detectors, it is now possible to explore and develop a reliable and highly accurate object detection and counting framework that can do this job, by simply using RGB images of the field under varying imaging conditions. However, a major limitation of such deep learning-based approaches is that they typically require a massive amount of hand-labeled images for training, which is still a tedious process. In this paper, we propose a label efficient deep learning framework for sorghum head detection and counting from UAV-based images. We first train a deep neural network based head detection model with a small number of labeled images. We then use this model to build a feedback loop to develop an automated framework that can generate
labels for new data (newly captured images) and feed it back to the neural network training process after the automated labels go through a significantly less tedious manual verification/modification step. Such training input help make the model better performing and robust through a quick and efficient retraining/fine-tuning process. We also visualize key features that the network learns from the training data-set that help the users better understand and trust the decisions that the trained deep learning framework makes. To demonstrate the performance of the proposed method, 1250+ manually labelled images were tested, with the mAP (mean average precision) of proposed method reaching a high 0.9413 (IoU threshold = 0.5), and the coefficient of determination, $R^2$ between human count and that counted by the proposed method reached 0.88, a significant increase over $R^2$ values reported in previous literature ($R^2$=0.56 and tested only on 40 images). The heritability of head counts obtained via the framework was 0.65 which is much higher than the heritability of head counts from manual counting (i.e., 0.31).

1.1.2 Deep Learning for exploring structure-property maps in photovoltaics

While in the previous chapter, we deal in generating a label-efficient automated framework, in this work, which is detailed in Chapter 4, we focus on building a strong trust model. We again take a real-world problem and aim to solve it. We first start by introducing a deep convolutional neural network (CNN) architecture that can serve as a fast and robust surrogate for the complex structure-property map. Several tests were performed to gain trust in this trained model. Then, we utilize this fast framework to perform robust microstructural design to enhance device performance. The Deep Learning model, termed DLSP (Deep Learning based Structure-Property exploration) is used as a substitute for the structure-property map to evaluate and quantify the performance of a particular microstructure. In order to build trust for the model, it is evaluated by visualizing the class specific outputs by using Saliency Maps. The model is also tested by using different test data-sets to infuse further trust and robustness.

The performance of an organic photovoltaic device is intricately connected to its active layer morphology. This connection between the active layer and device performance is very expensive
to evaluate, either experimentally or computationally. Hence, designing morphologies to achieve higher performances is non-trivial and often intractable. To solve this, we first introduce a deep convolutional neural network (CNN) architecture that can serve as a fast and robust surrogate for the complex structure-property map. Several tests were performed to gain trust in this trained model. Then, we utilize this fast framework to perform robust microstructural design to enhance device performance.

1.1.3 Top-K Feature Maps: Semi-supervised framework for explainable Deep Learning

Continuing the trust aspect put forth in Section 1.1.2, here we deal with keeping trust as well as extending this trust mechanism to develop an unsupervised framework to develop a crop disease quantification classification and quantification framework. Plant stress identification based on visual symptoms has predominately remained a manual exercise performed by trained pathologists, primarily due to the occurrence of confounding symptoms. However, the manual rating process is tedious, time-consuming and suffers from inter- and intra-rater variabilities. Our work resolves such issues via the concept of explainable deep machine learning to automate the process of plant stress identification, classification and quantification. We construct a very accurate model that can not only deliver trained pathologist-level performance, but can also explain which visual symptoms are used to make predictions. We demonstrate that our method is applicable to a large variety of biotic and abiotic stresses as well as is transferable to other imaging conditions and plants.

Current approaches for accurate identification, classification and quantification of biotic and abiotic stresses in crop research and production are predominantly visual and require specialized training. However, such techniques are hindered by subjectivity resulting from inter- and intra-rater cognitive variability. This translates to erroneous decisions and a significant waste of resources. Here, we demonstrate a machine learning framework’s ability to identify and classify a diverse set of foliar stresses in soybean [Glycine max (L.) Merr.] with remarkable accuracy. We also present an explanation mechanism using the top-K high-resolution feature maps that isolate the visual
symptoms used to make predictions. This unsupervised identification of visual symptoms provides a quantitative measure of stress severity, allowing for identification (type of foliar stress), classification (low, medium or high stress) and quantification (stress severity) in a single framework without detailed symptom annotation by experts. We reliably identified and classified several biotic (bacterial and fungal diseases) and abiotic (chemical injury and nutrient deficiency) stresses by learning from over 25,000 images. The learnt model is robust to input image perturbations, demonstrating viability for high throughput deployment. We also noticed that the learnt model appears to be agnostic to species, seemingly demonstrating an ability of transfer learning. The availability of an explainable model that can consistently, rapidly and accurately identify and quantify foliar stresses would have significant implications in scientific research, plant breeding and crop production. The trained model could be deployed in mobile platforms (e.g., unmanned air vehicles and automated ground scouts) for rapid, large-scale scouting or as a mobile application for real-time detection of stress by farmers and researchers. Further details are presented in Chapter 5.

1.1.4 Knowledge Embedding using Deep Learning

As our primary application, we design a model to generate solutions to partial differential equations, given their coefficients and boundary conditions. For physical systems that are completely described by their governing PDEs, the proposed architecture is entirely data-free, i.e., trained solely by enforcing the PDE as a structural invariance. We show that the model is also capable of conditional generation of solutions, providing flexible user control over both boundary conditions and PDE coefficients. As an example, we solve the classical non-linear time varying Burgers'equation (Bateman (1915)) in both viscid and inviscid cases, and demonstrate that the framework provides very competitive results. This marks a significant improvement over recently proposed unsupervised learning methods for solving PDEs, such as Raissi et al. (2017a,c); Zhu et al. (2019).

Finally, we also consider a more challenging application in computational material science, which is the problem of simulating (binary) microstructures obeying desired properties (Wodo and Ganapathysubramanian (2012a); Torquato (2013)). Here, the equations governing the formation of
microstructures are typically very complex, but we require that the target microstructures satisfy certain \textit{property-based} invariances. We encode these invariances in the form of moment matching constraints, and train an Surrogate-InvNet (S-InvNet) model to generate synthetic microstructures adhering to target statistics. We show that InvNet significantly outperforms standard numerical microstructure methods in terms of computational costs.

1.2 Physics augmented Deep Learning

Fully incorporating Physics and domain knowledge is an exciting new direction that is put forth as well. It proposes to incorporate and merge domain knowledge (models based on laws pertaining to and tailor-made for a specific application) with Deep Learning techniques, a concept that has not yet been explored in many scenarios and could prove to be exceptionally useful when dealing in vastly complex real-world problems with several layers of complexity. This approach can be application specific since such an idea will look at the task at hand and incorporate domain knowledge pertaining to that specific problem and utilize training data corresponding to the problem to develop the final model.

Although the idea of having a unique hybrid physics-driven DL model tailored for specific applications sounds promising, developing such models for every specific application can be extremely time and resource consuming. On the other hand, a unique idea that can be put forth here is whether an end-to-end framework can be developed where training data and physical laws can be fed into an universal model that can work pertaining to a class that handles a specific aspect of the problem at hand which can then be transferred to a similar aspect in another real-world problem. Thus, transfer-learning capabilities can also be induced to physics-driven DL frameworks, which might lead to significant time and resource savings.
Chapter 2. Preliminaries on Deep Learning and Training Considerations

This Chapter provides a general discussion on the different DL techniques devised to carry out our target objectives i.e., to perform the prediction, detection and classification tasks presented in subsequent chapters. Additionally, some training considerations taken into account while building and training such models have been discussed in this chapter.

This chapter is organized as follows: In section 2.1, we talk about Restricted Boltzmann machines, which are the fundamental units of Deep Neural Networks. In section 2.2, a general discussion of Deep Neural Networks is given, followed by a discussion of Convolutional Neural Networks in section 2.3. Then we discuss briefly on training considerations and existing methods to optimize neural networks. In sections 2.4 and 2.6, we discuss briefly on Auto-Encoders and Encoder-Decoder (ED) Networks.

2.1 Restricted Boltzmann Machines (RBMs)

A Restricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that learns features in an unsupervised manner based on a probabilistic model (Hinton and Salakhutdinov (2006)). It became popularized by Geoffrey Hinton in the mid-2000s (2006), when he and his group developed fast learning algorithms to effectively train RBMs (Hinton and Salakhutdinov (2006)). RBMs have been used in a diverse area of applications as a means to reduce data dimension (Van Der Maaten et al. (2009)), collaborative filtering (Salakhutdinov et al. (2007)), solving classification problems (Larochelle and Bengio (2008); Larochelle et al. (2012)), topic modeling (Xie et al. (2015); Hinton and Salakhutdinov (2009); Srivastava et al. (2013)) and feature learning (Boureau et al. (2008)). The success of RBMs largely owes to the fact that the extracted features is nonlinear in nature. As a result, it often generates good results when used in conjunction with a linear classifier.
such as a support vector machines (SVM) or a perceptron. An RBM basically attempts to maximize the likelihood of the data using a particular graphical model and typically employs the learning algorithm via stochastic maximum likelihood. Via this method, it is capable of capturing persistent regularities from the data and learning a probability distribution over the set of the provided inputs. However, a caveat is that, to effectively train an RBM model, we require a sufficiently large data set.

![Diagram of Boltzmann and Restricted Boltzmann Machines](image)

**Figure 2.1**: Differentiation between Boltzmann machines and Restricted Boltzmann machines (RBMs). In RBMs, there are no interconnections between the nodes in the same layer i.e., there are no visible-visible or hidden-hidden connections.

Restricted Boltzmann machines are actually a variant of Boltzmann machines (Ackley et al. (1985)). In a Boltzmann machine, nodes, or neurons in the same group are connected in addition to nodes from the other group (see Fig. 2.1). Typically, two groups are present in a Boltzmann machines: the visible and hidden units. However, the interconnectedness of all the neurons within and between groups may complicate modeling. Hence, a restricted version of the Boltzmann machine is used with a constraint that the neurons must form a bipartite graph where there are no connections within a group. This restriction make learning easier, as the hidden units become conditionally
independent given the visible states (Bengio et al. (2009)). RBMs are also the building blocks to deep neural networks (DNN), where they can be stacked to increase the modeling capacity.

### 2.2 Deep Neural Networks

Stacking multiple layers of RBMs together results in the class of architectures known as Deep Neural Networks. Fig. 2.2 illustrates this. Increasing the number of hidden layers in a network increases its nonlinear modeling capacity. Deep neural networks are also known as artificial neural networks (ANN), which are inspired by the observations and biological models proposed by Harvard neurophysiologists David H. Hubel and Torsten Wiesel. From these observations, they showed how the visual system of living animals builds complex representations from simple stimuli which established the fundamental concepts of deep learning (Hubel and Wiesel (1963); Weng et al. (1992, 1993); Riesenhuber and Poggio (1999)).

![Figure 2.2: Stacking of RBMs to form a Deep Neural Network](image)

In deep neural networks, each layer of the neurons trains on different sets of features using the outputs from the previous layer. The deeper we advance into the network, the higher the complexity of the features that the network can recognize. For example, in the application of face recognition, the first layer of the network usually learns primitives such as simple edges and curves. As we move on to the intermediate layers, the hidden layers begin learning a combination of these primitives,
such as the eyes, the mouth, the ears, and the nose. The deepest layers are capable of combining the parts and begin recognizing faces (see Fig. 2.2). This concept widely known as the hierarchy of features, that is, a hierarchy of features with increasing complexity and abstraction. Therefore, deep networks are suited for handling a large amount of very high-dimensional data sets.

### 2.3 Deep Convolutional Neural Networks (DCNNs)

DCNNs are suitable machine learning models because of their widely demonstrated efficacy in performing large-scale image classification, automated feature learning capability and ease of training (Krizhevsky et al. (2012)). Although the details of DCNN architecture and training procedures have been well described in recent literature (Krizhevsky et al. (2012); Lawrence et al. (1997)), we provide a brief description here for the sake of completeness. Compared with fully connected (FC) deep neural networks with the same number of hidden layers, DCNNs achieve a similar level of performance with fewer parameters to learn (Krizhevsky et al. (2012); LeCun and Bengio (1998)). DCNNs are designed to exploit the two-dimensional (2D) structure of an input image by preserving the locality of features via the utilization of spatially local correlations of an image by using tied weights, which are invariant to the translation of the feature positions (Krizhevsky et al. (2012); Lee et al. (2009)). Weight sharing among different locations in an image also increases the efficiency of learning because the number of learnable parameters during training are substantially fewer than those in an FC neural network.

In DCNNs, data are represented by multiple feature maps in each hidden layer. Feature maps are obtained by convolving the input image by using multiple filters in the corresponding hidden layer. In other words, they are obtained by repeatedly applying a function across sub-regions over the entire image, i.e., a convolution operation of the input image with a filter. To further decrease the dimension of the data, these feature maps typically undergo non-linear down-sampling with a 2X2 max-pooling operation (Boureau et al. (2010)). Max-pooling partitions (or super-pixelates) the input image into sets of non-overlapping rectangles and uses the maximum value for each partition as the output. Because neighboring pixels in an image share similar features, these pixels
can be discarded to overcome memory constraints and decrease training time. Furthermore, both spatial and feature abstractness can be increased by max-pooling, which results in increased position invariance for the filters (Boureau et al. (2010); Huang et al. (2007)).

To improve the performance of the architecture, a Batch Normalization layer is added between the two neuron layers, which normalizes the activations of the previous layer at each batch, i.e., applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1 (Ioffe and Szegedy (2015a)).

After max-pooling, multiple dimension-reduced vector representations of the input are acquired, and the process is repeated in the next layer to give a higher-level representation of the data. At the final pooling layer, the resultant outputs are linked to the FC layer, where Rectified Linear Unit
Figure 2.4: Illustration showing how 2D Max-Pooling works

(ReLU) activation outputs (Nair and Hinton (2010)) from the hidden units are joined to output units to infer a predicted class on the basis of the highest joint probability given the input data. With this setup, the probability of an input vector $v$ being a member of the class $i$ can be written as follows:

$$Pr(Y = i|v, W, b) = softmax_i(Wv + b) = \frac{e^{W_i v + b_i}}{\sum_j e^{W_j v + b_j}}$$  \hspace{1cm} (2.1)

where elements of $W$ denote the link weights and elements of $b$ denote the biases. The model prediction is the class with the highest probability:

$$y_{pred} = \argmax_i Pr(Y = i|v, W, b)$$  \hspace{1cm} (2.2)

The model weights, $W$, and biases, $b$, are optimized by the well-known error backpropagation algorithm (Rumelhart et al. (1985)), wherein true class labels are compared against the model prediction by using an error metric, which becomes the loss function for the (weights and biases)
optimization process. The loss function, chosen to be minimized for the data set \( V \), is the categorical cross-entropy function \((\text{Rubinstein (1999)}), \mathcal{L}\), and is given as follows:

\[
\mathcal{L}(V, Y) = -\frac{1}{n} \sum_{i=1}^{n} y^{(i)} \ln a(v^{(i)}) + (1 - y^{(i)}) \ln (1 - a(v^{(i)}))
\]  

(2.3)

Here, \( V = \{v^{(1)}, \ldots, v^{(n)}\} \) is the set of input examples in the training data set, and \( Y = \{y^{(1)}, \ldots, y^{(n)}\} \) is the corresponding set of labels for those input examples. The \( a(v) \) represents the output of the neural network given input \( v \).

### 2.3.1 Overview of the different layers used in DCNNs

In this section, a short description of the different layers used in developing and training a Deep Convolutional Neural Network model is presented. An illustration of a typical CAE framework is given in Fig. 2.5.

![Illustration of a typical Convolutional Neural Network architecture](image)

Figure 2.5: Illustration of a typical Convolutional Neural Network architecture

#### 2.3.1.1 The Convolutional layer

The function, “Convolution” is essentially a sliding function (a sort of filter) applied to a 2D or 3D input matrix. The layer associated with performing the convolution function is termed the “Convolutional Layer” and it plays the most important role in building and learning DCNNs. It
consists of filters (or kernels) that can be learnt through training. Such filters have small receptive fields to extract local structures and uses shared weights. The filters may specialize in detecting different features. The activations of different filters will depend on the specific type of feature at some spatial location in the input image. As the filters are convolved over the input, feature maps are generated (as illustrated in Fig. 2.3). Stacking the feature maps along the depth dimension gives rise to the full output volume of the convolutional layer.

There are three most prominent hyper-parameters that control the size of the output volume. With a deep layer, many neurons connect to a particular region in the input volume. The neurons will learn to activate different depending on the features in the input. For instance, neurons in the same layer may become active in the presence of edges in different orientations and lie in the same feature hierarchy. Large convolution strides will result in a smaller feature map due to less overlapping receptive fields, whereas small convolution strides (such as 1 X 1 → 1 unit along the horizontal and 1 unit along the vertical direction) convolution will result in strongly overlapping receptive fields and, subsequently, larger feature maps. In all cases, the dimensions will be reduced depending on the size of the convolution strides. However, it is sometimes desirable to preserve the original spatial dimensions of the input volume. To achieve this, the borders of the input volume can be padded with zeros (termed as zero-padding) such that after the convolution operation, the reduced spatial dimensions are compensated to maintain original dimensions.

2.3.1.2 The Pooling layer (max-pooling)

Pooling is a form of nonlinear downsampling. Common practices employ the maxpooling scheme (illustrated in Fig. 2.4), where a 4 X 4 matrix for example is downsampled into a 2 X 2 one by selecting the element with the highest value in the matrix. Other pooling functions exist too; one may downsample by averaging the values or even computing the \( l_2 \)-norm. Pooling is useful because it removes redundancies and helps reducing the dimensions of the data. The intuition is that once a feature is detected, the exact location of the feature may not be as important as the approximate location relative to other features. Doing so also reduces the number of learnable parameters to
combat overfitting as well as reducing computation time. Note that using pooling layers is up to the discretion of the user; it is a common practice to periodically insert a pooling layer after several convolutional layers. An additional benefit that pooling offers is the translation invariance of features. However, most studies are gravitating towards using smaller filters (Graham (2014)) and discarding the pooling layer (Springenberg et al. (2014)) in order to prevent an excessively aggressive reduction in dimension.

2.3.1.3 The Fully Connected (FC) layer

After several reductions in dimensions from convolution and pooling, the location information of the features become less important. Hence, we can connect the feature maps generated by the filters to the fully connected layers to increase modeling capacity. We can think of the feature maps being vectorized as an input to a single neural net layer. In the context of classification, a softmax function can be applied on the sigmoid activations of each output neuron to obtain a probability distribution, where the class with the highest probability is selected as the class prediction. Similarly, one can choose to minimize the loss function (such as negative log-likelihood) and optimize the model parameters via gradient descent.

2.4 Convolutional Autoencoders

An autoencoder tries to learn the approximation to the identity function in an unsupervised manner, such that the reconstruction of the input is similar to the actual input. In unsupervised learning, only unlabeled data is used. At first glance, it seems to be trivial to learn the identity function. However, this problem becomes not so trivial anymore if we impose some constraints to the learning process such that the algorithm can discover interesting or meaningful underlying patterns in the input data. An illustration of a typical CAE framework is given in Fig. 2.6.

For instance, in case of images, more specifically, in a natural 2D image are often correlated in terms of color. Objects in the image will also have a series of pixels that form the edges of the object. Autoencoders are algorithms that can automatically discover these correlations. Similar to principal
component analysis (PCA), autoencoders can learn low-dimensional representation of the inputs by capturing the codes within the data; in fact, the optimal solution to an autoencoder is strongly related to one found with PCA if linear activations or only a single sigmoid hidden layer are used (Bourlard and Kamp (1988)). The added advantage of autoencoders is that they can be stacked to form stacked autoencoders (SAE), another deep architecture that has a superior nonlinear modeling capacity compared to a single layer of autoencoder or PCA (Vincent et al. (2010)).

2.5 Training Considerations

2.5.1 Data Preprocessing

Data preprocessing plays an essential part before feeding input data into a neural network. Essentially a zero-mean data is desired as an input to a neural network, as it helps in faster model training and validation. For instance, consider what happens when the input to a neuron is always positive. In such a case, the gradients for updating the weights are always all positive or all negative. This calls for the data to be zero-mean. In practice, apart from making the data zero-mean and normalizing, there are other preprocessing techniques like PCA and Whitening. Fig. 2.7 illustrates a few data preprocessing schemes.
2.5.2 Batch Normalization

Introduced first in Ioffe and Szegedy (2015a), Batch Normalization is a form of regularization that in essence lessens the need for other forms of regularization in a neural network. The idea behind batch normalization is to normalize the inputs of each layer in such a way that they have a mean output activation of zero and standard deviation of one (i.e., make unit gaussian activations after each convolution or pooling layer before feeding the output to the next layer). This is analogous to how the inputs to networks are standardized. A known fact is that normalizing the inputs to a network helps it learn. But a network is just a series of layers, where the output of one layer becomes the input to the next. That means we can think of any layer in a neural network as the first layer of a smaller subsequent network. Thought of as a series of neural networks feeding into each other, normalizing the output of one layer before applying the activation function, and then feed it into the following layer (sub-network), helps the learning process for the network, easier, better and faster. Fig. 2.9 shows what Batch Normalization does to input data.
Figure 2.8: The idea behind performing Batch Normalization. *Illustration Courtesy: Stanford CS231n lecture notes*

### 2.5.3 Regularization: Dropout

Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. The term dropout refers to dropping out units (both hidden and visible) in a neural network. It refers to ignoring activation units (i.e. neurons) during the training phase of certain set of neurons, chosen at random. It was first introduced in Srivastava et al. (2014a). More technically, At each training stage, individual nodes are either dropped out of the net with probability \((1 - p)\) or kept with probability \(p\), so that a reduced network is left. Incoming and outgoing edges to a dropped-out node are also removed. This is essential in a network with a lot of learning parameters. In a neural network, a fully connected layer may occupy most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data. Dropout takes care of this issue.

### 2.5.4 Activation Functions

Fundamentally, neural networks are layered collections of nodes, each of which receives a set of inputs and individually makes the decision whether to fire, and propagate information downstream
to subsequent layers. The inputs are combined with weights and biases local to each node, which are updated by a learning algorithm in response to the observed error on training examples. This enables patterns in data to be learned by the neural network, where the value of the weight is proportional to its importance. The weighted input data from all sources is summed to produce a single value, (called the linear combination), which is then fed into an activation function that turns it into an output signal. Conceptually, the activation function is what makes decisions: when given weighted features from some data, it indicates whether or not the features are important enough to contribute to a classification. Hence the purpose of the activation function is to introduce non-linearity into the neural network. This allows the model to generate a response variable that varies non-linearly with its explanatory variables (one where the output could not be reproduced from a linear combination of inputs).

For our purposes, we used the Rectified Linear Unit (ReLU) function as the activation function of choice as it has significant advantages over other choices. It not only speeds up training but also for ReLU, the gradient computation is very simple. Computation step of a ReLU is easy as well. In the context of deep neural networks, the rectifier is an activation function, defined as:

$$f(x) = \max(0,x)$$  \hspace{1cm} (2.4)
where $x$ is the input to a neuron. A unit utilizing the rectifier is termed a rectified linear unit (ReLU).

### 2.6 Convolutional Encoder-Decoders in solving PDEs:

Very recently Bayesian deep convolutional encoder-decoders networks (a variation from the previously introduced Convolutional AutoEncoder, but acting on similar principles i.e., learning in an unsupervised manner) have been proposed to develop surrogate models for uncertainty quantification and propagation in problems governed by stochastic PDEs (Zhu and Zabaras (2018)) that uses a deep convolutional encoder-decoder network in a similar fashion to approaches considered in deep learning for image-to-image regression tasks. The proposed model acts as a surrogate by performing image-to-image regression using simulated output data to solve and perform uncertainty quantification tasks for such PDEs. As such, even though this method does not require label data, it does use simulated data to perform the regression tasks and does not make use of the underlying Physics that governs the PDE.

On the other hand, in Zhu et al. (2019), the authors introduce methodologies that does make use of the governing physics and at the same time uses a similar convolutional encoder-decoder structure. First, the authors show that when solving spatially varying PDEs (Darcy Flow), the Encoder-Decoder (ED) based parameterizations are computationally efficient in capturing multiscale features of the solution fields given a certain input field. They also show that ED-based method is more efficient than the FC-NN (Fully-Connected Neural Network) counterpart. Furthermore, we demonstrate that in comparison with image-to-image regression approaches that employ Deep NNs (Zhu and Zabaras (2018)), the proposed method achieves comparable predictive performance, despite the fact that it does not make use of any output simulation data. In addition, it produces better predictions under extrapolative conditions as when out-of-distribution test input data sets are used.
Figure 2.10: (a) A dense block contains $L = 3$ layers $h_1$, $h_2$, $h_3$ with growth rate $K = 2$. (b) The second layer $h_2$ of the dense block, where $x_2 = h_2([x_1, x_0])$ is its output feature map. Notice that the input to the third layer is the concatenation of the output and input features of $h_2$, i.e. $[x_2, x_1, x_0]$. As is often the case, each layer is composed of Batch Normalization, Rectified Linear Unit (ReLU) and Convolution (Conv). The convolution kernel has kernel size $k = 3$, stride $s = 1$ and zero padding $p = 1$, which keep the size of the feature maps the same as the input. Source and further details: (Zhu and Zabaras (2018))
Figure 2.11: Both (a) encoding layer and (b) decoding layer contain two convolutions. The first convolution reduces the number of feature maps while keeps their size the same using a kernel with parameters $k = 1, s = 1, p = 0$; the second convolution changes the size of the feature maps but not their number using a kernel $k = 3, s = 2, p = 1$. The main difference between (a) and (b) is in the type of the second convolution layer, which is Conv and ConvT respectively, for downsampling and upsampling. Note that no pooling is used at transition layers for maintaining the location information. The colored feature maps used here are independent from the feature maps with the same color shown in other figures. Source and further details: Zhu and Zabaras (2018)
CHAPTER 3. A WEAKLY SUPERVISED DEEP LEARNING FRAMEWORK FOR SORGHUM HEAD DETECTION AND COUNTING

In this chapter, we provide in-depth details on how we look at a particular use-case where the problem statement was to efficiently and accurately generate labelled data sets for training a DL framework. We eventually do this my implementing a semi-supervised HiL framework which is capable of doing just that in a self-sustained manner.

3.1 Introduction

Sorghum (*Sorghum bicolor L. Moench*) is a C4 tropical grass that plays an essential role in providing nutrition to humans and livestock, particularly in marginal rainfall environments. The timing of head development and the number of heads per unit area are key adaptation traits to consider in breeding programs, and this requires breeders to investigate and record the appearance of heads in the field plot by plot. However, for crops like sorghum, each planted seed can give rise to multiple tillers. Consequently, the number of heads per unit area can be greater than the number of plants, which dramatically increases the difficulties for such a survey, especially for large-scale breeding trails (viz., 100+ heads per plot times 1000+ plots, which is 100000+ when it comes to counting). So, in most cases, breeders will often investigate a part of the plot (Borrell et al. (2014)) (put a 1-meter stick randomly inside a plot and count the head number around it) or even skip this step to only measure grain size or weight from sub-sampled heads during harvest. Those alternative approaches fail to fairly represent variations within the entire plot, especially for breeding populations. This is especially important when it comes to providing information on the genotypic variation in the number of fertile heads produced per unit.

Machine learning and image processing have proved their utility in diverse fields. Especially in the field of plant phenotyping (Mochida et al. (2018); Zhang et al. (2017); Ghosal (2017);
Nagasubramanian et al. (2018); Ghosal (2017)), these tools have laid a strong foundation in detecting multiple crop diseases (Ghosal et al. (2018a)) as well as making sense of disease severity without the need for any additional human supervision (Ghosal et al. (2018a)), crop/weed discrimination (Guo et al. (2013); Lottes et al. (2017); dos Santos Ferreira et al. (2017); Louargant et al. (2018)), canopy/individual extraction (Varela et al. (2018); Mu et al. (2018)), fruit counting/flowering (Yamamoto et al. (2014); Guo et al. (2015); Sa et al. (2016)), head/ear/panicle counting (Madec et al. (2019); Hasan et al. (2018); Xiong et al. (2017)). Our hypothesis is that machine learning and image processing along with unmanned aerial vehicles (UAV) based photogrammetry is a reliable alternative to the labor-intensive sorghum head survey in the field (Sankaran et al. (2015); Singh et al. (2018a, 2016)). To the best of our knowledge, only one study has reported an approach to detect and count sorghum head from UAV images in recent literature (Guo et al. (2018)). The study uses carefully selected color and morphology features to train a two-step machine learning model to detect and count the number of heads from UAV images. It produces a good counting accuracy at the field level, but faces difficulties when it comes to the segmented single plot images because parts of the head along the plot boundaries have been cut out which leads to a loss in their morphological features.

Recently, performance of deep learning-based object detection has improved dramatically and has especially been useful for detecting partially occluded objects. However, as in case of most deep learning based algorithms, deep-learning models for object detection task require a massive amount of labeled images for training, in order to reach a desired level of accuracy. This again, is labor intensive and tedious. In order to overcome this limitation, researchers have proposed data augmentation (e.g., mirror-reverse, rotation, geometric transformation, scaling/zooming, contrast random transformation, random noise generation, random erasing of the original image) approaches to increase the number of training images Mikoajczyk and Grochowski (2018). However, in many cases, generic data augmentation fails and smart choices have to be made to augment the data-set for properly training the networkPerez and Wang (2017).
Figure 3.1: Web-based annotation tool used for labeling data. Here, the left panel shows labeling site, where labeling is done from scratch and the right panel shows validation site, where we validate the pre-labeled data (pre-labeled either by human annotators or the deep learning model)
In this work, we leverage another concept that is known as the *weakly supervised learning* to significantly reduce the cost of human labeling (Wea (); Perez et al. (2018)). There are several variants of weak supervision reported in literature. One such variant involves incomplete supervision (Zhou (2017)) where a small section of the training data is labeled and the remaining large section of data is unlabeled. In this situation, a machine learning (ML) model is usually trained partially with the labeled part of the data and then that semi-trained model is used on the unlabeled section of data to improve the model. There are again several possible strategies for such improvement. A simple strategy is known as self-training or bootstrapping (Biemann (2007)), that involves running inference with the semi-trained model for labeling the unlabeled data which can then be used for retraining the model. Note that such labeling could have a lot of inaccuracies, also known as noisy labels. Therefore, in order to function as intended, the initial semi-trained model needs to be sufficiently accurate. As this is a rather stringent requirement in practice, ML practitioners often involve an oracle (e.g., human annotator) to verify the annotations performed by the semi-trained model. However, rather than asking the oracle to verify/provide annotations for all unlabeled data, typically a much smaller number of *optimally* selected data samples are chosen for human labeling/verification. This class of weak supervision techniques is known as Active Learning (Huang et al. (2010); Settles (2009)), which has been widely studied in the context of shallow (Sivaraman and Trivedi (2010); Kutsuna et al. (2012)) and deep machine learning (Wang et al. (2017); Gal et al. (2017)). While this is the primary motivation of our proposed framework, we do not select such an optimal subset of data samples for human verification. Instead, we select a random subset of data samples and present their labels (generated by the semi-trained model) for human verification. Therefore, we simply refer to our proposed scheme as a weakly supervised deep learning framework. Note that one can also avoid involvement of human annotators altogether via employing semi-supervised techniques (Zhu (2005); Chapelle et al. (2009)). We compare the outcomes of the proposed framework with both human annotation of the images as well as human-counting of sorghum heads in the field.
Note, due to the size of the experiment-field, field counting by human counters was conducted with a sub-sampling strategy, that is the current standard in breeding. ¹

### 3.2 Field experiment, Data Collection and Preparation

The field experiment was conducted at Hermitage, Queensland, Australia (28.21°S, 152.10°E, 459 m above sea level) during the 2015-16 summer growing season and the seeds were sowed on 22nd December, 2015. The detailed information for field configuration and UAV flight design is described in our previous work Guo et al. (2018). From 2109 originally collected UAV images (resolution 5472×3648), 3D point cloud and orthomosaic images were generated, which were then used to extract the individual plots using our previously reported method Duan et al. (2016); Guo et al. (2018). Average plant density was 115,000 plants per hectare. Each plot consisted of two 5m long rows. The trial used a solid row configuration with a row spacing of 0.76 m while the distance between two neighboring plots was 1 m. In total, 28825 individual plot images were extracted from original images with specific rotations applied for generating varying field orientations. There were a total of 1440 plots laid out as 36 columns × 40 rows with several columns (216 plots in total) being filler plots for spraying access. These 1440 images are a subset of the initially extracted 28825 images. Out of those 1440 plot images, 1269 were labeled.

We also use two separate data-sets ('40Plots' and '52Croppped') from our previous work (Guo et al. (2018)) to compare the performance of the proposed method with the state-of-the-art results, reported there.

A web based interactive labeling tool (see Fig. 3.1) (Web Tool for Annotation) was designed to label the rectangular regions of interest (ROIs) of ground truth data from images. The tool allows single or multiple human annotators to label a same set of image data at the same time. Furthermore, all labeled data are subsequently validated by an administrator (who also serves as an expert in the field) to ensure the quality. The tool then exports the labeled data with a KITTI format text file. The validation protocol works as follows - first, a labeler works on the data set and

¹This strategy involves placing a 1-meter stick randomly within a plot and counting the sorghum heads around it. This is done by breeders in the field for each plot.
3.3 Methods

3.3.1 Deep Learning Framework

We devise a RetinaNet (Lin et al. (2018)) based approach with the residual network, ResNet-50He et al. (2016) as our backbone architecture. Focal LossLin et al. (2018) has been used as the loss function of choice for training the sorghum head detection network.

RetinaNet and Focal Loss have been well described in Lin et al. (2018), but we discuss the same here briefly for the sake of completeness. RetinaNet is devised as a network consisting of a “backbone” network and two sub-networks (subnets) with specific functionalities. The former computes a convolutional feature map (F.M) for a given input image and in our case, this “backbone” network is simply a convolutional (residual) network ResNet-50. After this feature map is computed,
the first sub-network performs convolutional object classification on the output that the residual
network (ResNet-50) produces while the second subnet performs convolutional bounding box
regression. The two subnets thus work in tandem and are specifically tailored and suitable for
single-shot, dense detection problems. This unique capability of RetinaNet makes it especially
suitable for our problem considering, we have image samples that have many target objects in each
(both overlapping and non-overlapping) for detection. We leverage this dense property of our data
set and use this specific method as a backbone for developing our framework.

3.3.1.1 Backbone Network

The Feature Pyramid Network (FPN) (Lin et al. (2017)) is adopted as the backbone network.
FPNs are effective in constructing rich, multi-scale feature pyramid (and generate several layered
representations of features from target layers in the neural network) from an input image. From
these pyramids that the FPN presents, every level of the pyramid can be queried for detecting
objects at different scales. This improves multi-scale predictions from fully convolutional networks.
In our case, we build a FPN on top of the ResNet-50. We use pyramids with levels $P_3$ through $P_7$
(the default setting reported in Lin et al. (2018)). A point to note here is that pyramid, $P_s$ has
resolution $2^s$ lower than the input, where $s$ indicates pyramid level. All the pyramid levels have 256
channels. Details of the pyramid are in Lin et al. (2017).

3.3.1.2 Regression Subnetwork

For the Regression subnet, anchor boxes similar to those in Lin et al. (2017) were used. The
lower limit on the area of the anchor boxes is $32^2$ and the upper limit is $512^2$ for pyramid levels
$P_3$ to $P_7$, respectively. These anchor boxes are also translation invariant. Details of the anchor
design are mentioned in Lin et al. (2017, 2018). It is worth mentioning here that each anchor is
assigned a length $K$ one-hot vector of classification targets (for our case, $K = 1$ since we only
have one class, i.e., Sorghum Head (SH)) and a 4-vector of box regression targets. We use an
intersection-over-union (IoU) threshold of 0.5 to assign the anchors to ground-truth object boxes.
Figure 3.3: Automated annotation protocol: The initial input image passed through the network is part of the training data-set. This trains the RetinaNet architecture and generates a model, which then is used to annotate new data. This annotation is then verified by human expert raters and fed back into the network as new training data. Process is repeated until network reaches desired level of accuracy.
and reject assigning anchors if their IoU is in the range $[0, 0.5)$. We carry out a few experiments by varying the IoU threshold from 0.4 to 0.9 to see how the network performs in terms of mAP values. This is reported in Section 3.5. However, for most applications as well as for ours, an IoU threshold of 0.5 is acceptable.

### 3.3.1.3 Classification and Prediction

To predict the probability of object presence at each spatial position for each of the anchors and the object class, we use a classification sub-network (subnet). Since we have only one class (label), for prediction purposes, a simple residual network, ResNet-50 is sufficient. The classification subnet is attached to each Feature Pyramid Network (FPN) level with its parameters sharing across the different levels of the pyramid network. An input feature consisting of $C$ (here, we have $C = 256$) channels from a target pyramid level is taken and four $3 \times 3$ convolutional (conv) layers each with $C$ filters are applied, with ReLU activations on top. After that, another $3 \times 3$ conv layer with $A$ (here, we have $A = 9$) filters is added. We then apply sigmoid activations to get a binary prediction per spatial location. Note that, no parameters are shared between the box regression subnet (which we describe briefly in the next paragraph) and this particular object classification subnet.

### 3.3.1.4 Bounding Box Regression

Parallel to above-mentioned classification subnet, the box regression network is used to regress the offset from each anchor box to a nearby ground-truth object. A small-scale fully-connected network is coupled with each pyramid level to carry out the regression task. The box regression subnet shares an identical design with the classification subnet with the only exception that it generates $4A$ linear outputs for every spatial location. Thus, for every $A$ anchors for each spatial location, these four outputs help predict the relative offset between the ground-truth box and the anchors. The bounding box regressor used in this case is class-agnostic.
3.3.1.5 Data Description and Loss Function

We initially collected 283 hand-labeled canopy images. Out of that initial pool of 283 images each constituting of 2 rows of sorghum plants, our training data includes a randomly selected subset of 40 images. This choice of 40 images is further discussed in Section 3.4. The network was initialized with random weights and the training was carried out for 40 epochs with a batch size of 10 to generate the model.

Regarding the choice of the Loss Function, we use Focal Loss (Lin et al. (2018)). Focal Loss (Lin et al. (2018)) extends the Categorical Crossentropy (CE) loss for binary classification as follows:

- First, CE loss is defined as:

\[
CE(\alpha, x) = \begin{cases} 
-log(\alpha) & \text{if } x = 1 \\
-log(1 - \alpha) & \text{otherwise} 
\end{cases}
\]

where \(x\) is the ground-truth class and \(\alpha \in [0, 1]\) is the probability estimated by the model for class with label \(x = 1\). To compress this CE loss definition, we further define:

\[
\alpha_t = \begin{cases} 
\alpha & \text{if } x = 1 \\
1 - \alpha & \text{otherwise} 
\end{cases}
\]

Thus, \(CE(\alpha, x) = CE(\alpha_t) = -\log(\alpha_t)\). Now, Focal Loss (FL) is defined as:

\[
FL(\alpha_t) = -(1 - \alpha_t)^{\tau}\log(\alpha_t) \quad (3.1)
\]

where, \((1 - \alpha_t)\) is a modulating factor added to the cross entropy loss with \(\tau \geq 0\) being a hyper-parameter termed the focusing factor. Further, a balancing factor, \(\gamma\) is added to the FL term to yield slightly better accuracy:

\[
FL(\alpha_t) = -\gamma(1 - \alpha_t)^{\tau}\log(\alpha_t) \quad (3.2)
\]
Adam (Kingma and Ba (2014b)), with a learning rate of 0.00001 was used as the optimizer of choice.

### 3.3.1.6 Training hardware, source code and libraries used

We train our network using a NVIDIA Tesla P40 GPU (22GB of GPU memory) while evaluation and inference were carried out using a NVIDIA GeForce GTX 1070 (8GB of GPU memory). Training, evaluation and testing were done using the Keras Deep Learning library with Tensorflow backend on Python 2.7. The source code on RetinaNet built for Keras was accessed from Ker() and then modified for our problem. The codes used for generating results and reproducing the results presented in this work are available at DeepSorghumHead. ²

### 3.3.2 Reducing human labeling cost via a weak supervision strategy

Here we describe our proposed weak supervision protocol as illustrated in Fig. 3.3. A brief overview of deploying and inner workings of our framework for carrying out the weakly supervised training can be stated as follows: we first start with an initial model trained with a single image (randomly chosen from the pool of 283 images). Upon successful training (all possible hyperparameters optimized as much as possible), we extract a trained model at epoch 40 of the training step. Considering we have only one image, our initial training step is quite fast with each epoch taking about 200s. Following our protocol illustrated in Fig. 3.3, we then utilize this semi-trained model to generate the labels for another randomly selected image from our 1440-image data set. The generated labels and the image itself are then fed into the image annotator app for validating the bounding box locations and corrections are made by a human annotator if and as necessary. The corrected labels are then extracted as new labels for the image and we then add this labeled image to our training set. Our training data set (now consisting of two labeled images) is then used to re-train the deep learning model to build an improved version that performs better than our initial model. We iterate this process over our pool of training samples until we reach our desired performance as determined by evaluating the trained model at the end of every iteration. At the

²https://github.com/oceam/DeepSorghumHead
end of this process, we obtain our final model which then can be deployed in real time to predict crop (sorghum)-yield and estimate productivity.

As discussed in the introduction, our primary motivation for introducing this kind of strategy is active learning. However, we stop short of calling our framework that and simply refer to it as a weak supervision strategy instead, due the rationale presented below. In the current context, the goal of an active learning framework would be to leverage the semi-trained learning model (trained with a small subset of labeled data) to optimally select a subset of (originally) unlabeled training data samples for human labeling/verification. However, in this work, we just randomly select such a subset and not use any optimal selection procedure. The primary objective here is to show a proof-of-concept that such synthetic annotation can dramatically reduce the cost of human labeling. In our case, we see, approximately a four-fold reduction in annotation time when a human just verifies/modifies synthetically annotated data by a semi-trained model (see Fig. 3.5). However, it is to be noted that framing this problem rigorously as an active learning problem remains a key future research focus for us.

3.3.3 Evaluation Metrics

We use Pearson’s Linear Correlation Coefficient, the Coefficient of Determination ($R^2$) and mAP (mean average precision) based on an IoU (Intersection Over Union) threshold. We also calculate and compare heritability ($h^2$). These metrics, although well defined in literature, are briefly discussed here for the sake of completeness.

3.3.3.1 Pearson’s Linear Correlation Coefficient ($\rho$) and Coefficient of Determination ($R^2$):

Given a pair of random variables $(X, Y)$, Pearson’s Linear correlation coefficient is defined as:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$$  \hspace{1cm} (3.3)
where \( \text{cov}(X,Y) \) denotes the covariance between \( X \) and \( Y \) and \( \sigma_X \) and \( \sigma_Y \) denote the standard deviation of \( X \) and \( Y \) respectively. This can be extended to paired data, \( X : (X_1, X_2, ..., X_n) \) and \( Y : (Y_1, Y_2, ..., Y_n) \) consisting of \( n \) pairs. In that case, we have,

\[
\rho = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2}}
\] (3.4)

where \( n \) is the sample size, \( x_i, y_i \) are the individual sample points indexed by \( i \) and \( \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \) and \( \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \) are the sample means for \( X \) and \( Y \) respectively. The correlation coefficient, \( \rho \) can range from -1 to +1. A value of +1 implies that there exists a perfect positive linear correlation between \( X \) and \( Y \) i.e., as \( X \) increases, so does \( Y \) following the perfectly positive correlation for all existing points. A value of -1 signifies perfect negative linear correlation i.e., \( X \) increases, \( Y \) decreases following the same perfectly negative correlation for all existing points. A zero correlation indicates there is no linear correlation between \( X \) and \( Y \). For additional information on \( \rho \), refer to Cowan (1998).

Now, the Coefficient of Determination, \( R^2 \) is simply the square of the linear correlation coefficient i.e., \( R^2 = \rho^2 \). \( R^2 \) thus takes values between 0 and 1. For additional information on \( R^2 \), refer to Cowan (1998).

3.3.3.2 Mean Average Precision (mAP):

To provide a brief overview of the mAP measure, we first review a few basic metrics such as True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) measures.

Consider a binary (0 - negative or 1 - positive) classification problem. Results that fall under TP are those samples that are truly 1 and have been correctly classified by the Machine Learning (ML)/ Deep Learning (DL) model as 1. Results that fall under TN are those samples that are truly 0 and have been correctly classified by the ML/DL model as 0. Results that fall under FP are those samples that are truly 0 and have been incorrectly classified by the ML/DL model as 1 and results that fall under FN are those samples that are truly 1 and have been incorrectly classified by the ML/DL model as 0.
Now, we define two more measures - Precision and Recall. Precision is defined as:

\[
\text{Precision} = \frac{TP}{TP + FP}
\]  

(3.5)

while Recall is defined as:

\[
\text{Recall} = \frac{TP}{TP + FN}
\]  

(3.6)

Upon defining these two measures, we now define Average Precision (AP). For a ranked sequence of outputs, precision varies with recall and a Precision-Recall curve can be plotted. For precision, \( p(r) \) as a function of recall \( r \), the average precision is defined as the area under the P-R curve and is given by:

\[
AP = \int_{0}^{1} p(r) \, dr
\]

(3.7)

For the discrete cases (which is most practical cases) we have,

\[
AP = \sum_{j=1}^{1} P(j) \Delta r(j)
\]

(3.8)

where \( j \) is the rank in the sequence of samples, \( n \) is the total number of samples and \( P(j) \) is the precision at the cut-off point, \( k \). \( \Delta r(j) \) is change in recall from \( j - 1 \) to \( j \). Once we define Average Precision (AP), \( mAP \) for a set of queries is simply defined as the mean of the APs over those queries. Thus, for \( q \) queries, average precision as a function of those queries, \( AP(q) \) and \( Q \) being the total number of queries, we have:

\[
mAP = \frac{1}{Q} \sum_{q=1}^{Q} AP(q)
\]

(3.9)

For additional information on \( mAP \), refer to Manning et al. (2010).

3.3.3.3 Intersection over Union (IoU):

Also termed Jaccard Index, we define this metric as follows:
where $A_o$ is the area of overlap and $A_u$ is the area of union of the two sets. In Fig. A, we represent the sets as squares for the sake of simplicity. For our problem, we report our results considering a IoU threshold of 0.5. We also explore other IoU values ranging from 0.4 to 0.9 and report corresponding mAP values in the discussion section.

For additional information on IoU, refer to this source IoU ().

### 3.3.3.4 Heritability of breeding population:

Plant breeders use the term heritability (or repeatability) as a measure of how much of the variance in a trait (such as head number per plant) is associated with genetic rather than random environmental factors Falconer et al. (1996). This statistic can be estimated whenever sets of genotypes are compared (with some degree of environmental replication). If a trait can be measured with high heritability, it means that the breeder can have greater confidence that selecting for the trait is associated with a true genetic rather than environmental or experimental effect. Hence, heritability is a useful statistic to compute in such populations, in addition to standard statistics related to correlation, precision and recall.

For example, heritability was used to evaluate the performance consistency of methods for estimation of plant height Hu et al. (2018). It indicates the proportion of total variation of a trait that is due to differences between genotypic individuals M. R. (2002). In the early generation of breeding population, simple traits with high heritability are selected to rapidly fix the associated genes and genomic regions Mace and Jordan (2010). In this work, we calculate and compare the
heritability of head number from human field counting and the proposed method as well. In the experiment we have done here, the breeding population is at an early generation (still quite diverse and large) and so the estimates of heritability are pertinent to decisions for the plant breeder on whether to use head number or not as a selection trait. Until the development of image-analysis methods, it has been labor-intensive to measure head number, so there is little data available on its heritability.

Formally, we define heritability using a linear mixed model that was designed to estimate the variation components via the restricted log likelihood (REML) method with the R package SpATSVelazco et al. (2017). The spatial trend is counted by a two dimensional P-spline (i.e. row and column in the experimental design). Genotype and replicate effects were considered as random factors. With this setup, heritability is calculated as:

\[ h^2 = \frac{V_g}{V_t} \]  

(3.11)

where \( h^2 \) is heritability, \( V_g \) is the genotypic variation component and \( V_t \) is the total variation component.

3.4 Results

3.4.1 Weakly supervised deep learning model training

Following our protocol as described in the first paragraph of Section 3.3.2 (see Fig. 3.3), we observe that \( R^2 \) value for 2-image model is significantly greater than the \( R^2 \) value for the first-step single-image model (see Fig. 3.4). We keep iterating until we reach a high \( R^2 \) value of 0.8815 for the model trained with 40 randomly selected images, after which the \( R^2 \) value decreases slightly for the 50 image model. For the set of all 283 images that we set aside initially as our training set, we see \( R^2 = 0.8797 \) (which is still slightly lower than \( R^2 \) for the 40-image trained model). In Fig. 3.5, we show the model output for an image tested with the 5-image trained model (Fig. 3.5 (A)) and compare it with the manually verified/corrected image (Fig. 3.5 (B)). The true label count
Figure 3.4: Variation of $R^2$ for the 1260-test data set with change in training data size (following our automated annotation protocol (see Fig. 3.3), varying from models trained with 1, 2, 3, 4, 5, 10, 20, 30, 40, 50 and 283 images.
for the image was 104 sorghum heads which took about 720 seconds to be manually labeled from scratch. On the other hand, using the 5-image model, the corrections that had to be made were only for six over-detected sorghum heads and and three undetected heads. Typically, over-detection happens only when there are multiple sorghum heads in close proximity - which drives the model to predict more heads at such locations than the actual number. Missed detection can be attributed to the significantly low number of training samples (five samples only in this case). Making these few corrections took about 180 seconds. In majority of the cases, when the model output images are fed back into the image annotator app, we note about 4 times speed up on an average as compared to the manual annotation process from scratch. Evidently, this depends on how fast the human expert performs the annotations and subsequent validations. The quality of the semi-trained model being used to generate the labels in the first place is also a critical factor. Thus, this kind of approach provides a significant advantage in terms of being a lot less expensive and more time-efficient without sacrificing performance.

Fig. 3.6 shows qualitative comparison between the models trained with varying number of training samples, starting from 1-image to 5-images with increment of 1 and then from 10 images to 50 images with increment of 10 and finally using all the available 283 training images at our disposal. We randomly select an image from our 1260-image test data set and get the outputs for different models. Clearly we see here that the 1-image model shows the poorest performance ($R^2 = 0.3549$, see Fig. 3.4) with several sorghum heads missed. As we progress further, we see that the number of missed detections falling down to just 2 for the 5-image trained model ($R^2 = 0.8163$, see Fig. 3.4) and then finally to 1 for the rest of the models (trained with 10, 20, 30, 40, 50 and 283 images). We observe that the one sorghum head that remains undetected is in extreme close proximity to another head. Hence, accurate counting in this case becomes difficult even for human experts. The other problem that arises when following the automated annotation protocol, is over-detection of sorghum heads, which also happens when multiple heads are in close proximity. From Fig. 3.6, we note that over-detections are there even for the model trained with all available training data (283 images), although much less compared to the earlier models (1,2,3 image based models for instance).
Figure 3.5: (A) Model output image (about 180 seconds for verifying and making corrections) vs (B) Manually annotated image starting from scratch (about 720 seconds to label the entire image)
Figure 3.6: Comparing results for a randomly selected test sample (genotype G357) for different models - starting from model trained with one sample and passed through our weak supervision based learning protocol using 2, 3, 4, 5, 10, 20, 30, 40, 50 and 283 training samples.
3.4.2 Evaluating the proposed framework

We initially set out to test our model on our test data set consisting of 1269 hand-labeled images. However, out of those 1269 images, nine images have white sorghum heads that were not represented at all in the training set. Hence, we do not consider those images for calculating the $\rho$ and $R^2$ values and report metrics based on only 1260 test images. Fig. ?? and Fig. 3.12 show the detection results. The detected sorghum heads are shown within red bounding boxes. It is observed that the model effectively detects all the sorghum heads across different genotypes - G96 and G357 (captured in bright and dark conditions). The coefficient of determination ($R^2$) between the manual and proposed method of counting achieved is 0.8815. This implies that the model performs quite well in estimating the sorghum head counts, which in turn, leads to getting accurate estimates of overall crop yield. The manual counting is a two-pronged approach - first, from the RGB images, a manual labeling of all distinctly visible heads was done such that the number of labeled bounding boxes is same as the manual count from the image. The second method involves a field count where, in the field, for each plot, breeders put a 1-meter stick randomly inside a plot and counted the number of sorghum heads around it. The manual count from the RGB images are used to calculate the correlation plot, shown in Fig. 3.7. On the other hand, the manual count from the field is used to calculate heritability, as reported in the next paragraph. We however, do not consider calculating the count correlation between the field count and that based our proposed approach. Note that we also use two separate data sets (‘52Cropped’ and ‘40Plots’) from our previous work (Guo et al. (2018)) in Fig. 3.7.

The heritabilities were 0.31 and 0.65 for head numbers by human field counting (counted heads in a 1 linear meter per plot) and by the proposed method (counted heads for entire plot), respectively. A higher heritability indicates that the proposed method is able to capture the genotypic variations in the field from where the data was collected. This higher heritability indicates that the proposed method is more reliable, and the counted head number could be used as a selection trait for breeding experiments.
Figure 3.7: Correlation plots for: Test 1260-image test data-set (left) with $\rho = 0.9389$, $R^2 = 0.8815$ @ mAP of 0.9413 (IoU threshold = 0.5), test data-set labeled “52Cropped” (right-top) with $\rho = 0.9063$, $R^2 = 0.8214$ and test data-set labeled “40Plots” (right-bottom) with $\rho = 0.8756$, $R^2 = 0.7667$. 
3.5 Discussion

The model is observed to be robust and at the same time sufficiently accurate in detecting the sorghum heads leading to a good estimate of the crop yield. The model also helps in future annotations of new data by drastically reducing the time taken to annotate new data-sets with numerous target annotations, which can be fed back to the neural network to further improve the model.

A key point to note in Fig. 3.7 is that for most cases, the model overestimates the predicted counts. For most of these cases, the overestimate is by 5 or 10 counts only for the 1260-image and the “40Plots” test set. This overestimate is a result of some of the heads being very close to each other and overlap, which results in the model detecting more heads than present in those regions. In case of the “52Cropped” data-set, the model underestimates the count due to the presence of white-sorghum heads.

3.5.1 Comparison with state-of-the-art:

We noticed a significant improvement over the previous known best method for counting. In the previous method Guo et al. (2018), the authors test their method on the “52Cropped” and the “40Plots” data-set where they achieve a $R^2$ value of 0.84 and 0.56 respectively, whereas our method achieves 0.82 and 0.77 $R^2$ values on those data-sets. Here we note that our model performs just as well for the “52Cropped” data-set and significantly better for the “40Plots” data-set. However, even for the “52Cropped” data-set, where $R^2$ value for our method is only less by 0.02 compared to the previous best (Guo et al. (2018)), we note that this is because this particular data-set has images having a significant number of white-sorghum heads in them, and as we will discuss in the Failures Cases section, there were no samples of white-sorghum heads in our training data-set. Even then, our model was able to detect a good number of these heads and achieve similar level performance with the previous best. This leads us to propose that adding white sorghum head data to the already existing training data-set would significantly increase the $R^2$ values for this particular
data-set as well. This is left as a future direction and continuation of this work. An example output for a sample image from the “52Cropped” data-set is shown in Fig. 7.

3.5.2 Choice of Training Data size:

We initially collected a data-set consisting of 283 hand-labelled images. Upon exploring various training data-sizes that can efficiently and robustly train the neural network, we chose a data-set of 40 images out of those 283 for training. The $R^2$ values were the highest for this data-size ($R^2 = 0.88$). We calculated $R^2$ values for 30-image data-set as well as 50-image set which gave us $R^2$ values of 0.87 and 0.88 respectively.
Figure 3.9: Feature Map Visualization for test image from genotype G40
3.5.3 Visualizing learnt features:

We prune target layers of the neural network to see what features are learnt hierarchically as the image is passed through during the testing phase. Fig. 3.9 shows such an example. As we can see, the residual network is able to learn the sorghum head shapes and is able to clearly catch the different head sizes as well. We put forth this visualization as a trust mechanism that gives the reader information on whether the neural network actually learns important visual cues that humans usually tend to look for when doing manual surveys in the field. It turns out that deep neural networks do that as well as can be seen from Fig. 3.9, where we take a random test sample and visualize all the 64 feature maps (FMs) from the 3rd layer of the network and choose top 4 from among them that actually inspect the sorghum heads. A few more examples are presented in Fig. 3.10.

3.5.4 Robustness to varying orientations:

The model is also robust to varying orientations and augmentations of the plot images. This is a very interesting aspect as in most practical cases (e.g., when deploying the model on drones or on ground robots), it is not always possible to take perfectly vertically oriented images or videos of the plots that match those in our training data set. Thus, being robust to augmentations even when trained on handpicked images (that represent about 2 vertical crop columns each) is an extremely desirable quality for such a framework (see Fig. 3.11).

3.5.5 Choice of IoU threshold:

As seen in Fig. 3.14, we chose an optimum IoU threshold of 0.5, as anything less than that gives us additional bounding boxes for every detected sorghum head and affects the mAP values. Changing the IoU threshold, however, does not affect the $R^2$ value and it remains constant at 0.88
Figure 3.10: Feature Map Visualization for brown sorghum heads (top) and white sorghum heads (bottom)
Figure 3.11: Model output for varying orientations. Here we take the original image from the test set and perform several counterclockwise (CCW) rotations and then test our model on these images. As can be seen from the detection results, the model accurately picks up on all the sorghum heads in the original as well as the transformed images.

for our 1260-image test-set. Thus, for most cases, choosing \( \text{IoU} \geq 0.5 \) is a standard practice and we follow this.

### 3.5.6 Unseen Cases:

The model fails in cases where the sorghum heads are very small (less than 32 area pixels in size) and in cases where a completely different genotype depicting white sorghum heads are present. While the case for small sorghum heads is taken care of by resizing the image to an appropriate size where no sorghum-head shape go below the lower limit of 32-area pixel area, the genotype issue can only be addressed by including such examples into the training data set. This genotype is identified as G286. The model fails to detect these white sorghum heads in the first place, as there were no such training examples in the training data set to start with. Fig. 3.15 shows such an example. We also visualize the learned features for two such white-head samples to see what the network looks at for these cases. This is shown in Fig. 3.16 (right). A point to note here is that, even though the network had no white-head training samples to begin with, it can capture the shape-size features for some of these heads, but then fails for others.
Figure 3.12: More detection results for Sorghum Plot images across different genotypes and variations.
Figure 3.13: Variation of $R^2$ for the 1260-test data set with change in orientation. The $R^2$ values are for the trained model (trained with 40 images)
Figure 3.14: The variation of mAP with respect to IoU shows that mAP begins to drop drastically after the 0.6 IoU threshold. Thus, for simplicity and getting satisfactory outputs from the model, an IoU threshold of 0.5 is chosen, which is a widely accepted threshold in the deep learning and object detection/localization community.

Figure 3.15: Unseen Case: Genotype G286 (white-heads). Although the model had no white-head samples to train on, it is still able to detect some as seen in the sample to the right.
Figure 3.16: Feature Map Visualization for brown sorghum head (left) and white sorghum head (right). For the brown sorghum head, we see that framework specifically picks up the region around the heads while it does not do the same for the white heads.

3.5.7 Concluding Remarks:

A few concluding statements that we draw from our work is summarized as follows:

- Proposed label-efficient re-learning strategy significantly increased the efficiency of training data collection and annotation, which otherwise has been a tedious and time-consuming process,
- Very few hand-labelled data (40 images) were required to achieve desirable results,
- Our proposed framework is robust to varying orientations as well as different lighting conditions,
- A high counting accuracy ($R^2 = 0.88$ for plot level) at a high $mAP$ of 0.94 with $IoU \geq 0.5$ was achieved on our 1260-image test set,
- Proposed a direction for making deep-learning based object detection in the agriculture domain understandable/explainable and trustworthy,
• An interesting observation through this work was that for object detection in the agriculture domain, labeling similar images does not increase the accuracy of the model.
CHAPTER 4. INTERPRETABLE DEEP LEARNING FOR GUIDED STRUCTURE - PROPERTY EXPLORATIONS IN PHOTOVOLTAICS

In this chapter we develop a classification framework as a structure-property mapping and subsequently explore trust generation for the developed model from an interpretability point of view. Subsequent sections further expand on this.

4.1 Introduction

Mapping microstructure to macro-scale property of materials and controlling such relationships has been an important theme of modern materials research (Ju et al. (2017)). Despite the investment of millions of research hours and dollars (Foundation ()), this largely remains elusive to the research community. A major reason is that the map from microstructure (domain) to property (co-domain) is inherently non-surjective and highly non-linear, making incremental judgments based on incremental modifications infeasible. Added to this is the infinite dimensionality of microstructure space, which makes the description of the domain in-exhaustible. Hence, estimating the property of one morphology from another visually similar morphology is not trivial – each morphology will require a full scale experiment/simulation for reliable quantification.

Modern engineering applications are driving the demand for heterogeneous materials with tailored multifunctional properties. Very often, these properties are dependent on the microstructure. In recent years, there has been a sustained focus on microstructure-sensitive design. The design intent here is to identify tailored microstructures that result in desired properties.

The rational design of heterogeneous materials has emerged as a very promising approach towards discovery of new materials and devices with tailored properties and subsequently spur novel applications. One such application example has been that of organic electronics, specifically organic photovoltaics (OPV). In spite of exhibiting multiple benefits (tunability, flexibility, cost,
low-temperature manufacturability), organic photovoltaic films still remains a niche market due to relatively poor photoconversion efficiency compared to inorganic counterparts. Careful theoretical (Kodali and Ganapathysubramanian (2012b); Wodo et al. (2012); Casalegno et al. (2010); Meng et al. (2009); Marsh et al. (2007); Watkins et al. (2005)) and experimental analysis (Marsh et al. (2010); Hwang et al. (2008); Hoppe and Sariciftci (2004); Giridharagopal et al. (2010)) have revealed how the microstructure impacts each stage of the photo-conversion process. However, the complexity of these analysis approaches have made systematic exploration infeasible, with the result that there exist no design principles nor approaches for identifying promising microstructure in a systematic way. Thus, a key bottleneck to microstructure sensitive design is the paucity of techniques that can rapidly evaluate the performance of a microstructure.

Our approach to resolve this bottleneck is through machine learning (ML), which is used to create a fast surrogate for any complex functional map in a data-driven manner. Over the last decade, machine learning models have proved their ability to ingest volumes of data-label pairs and create efficient proxy or surrogate models to predict labels for similar instances of data. Deep Learning, the state-of-the-art ML form, has especially advanced the field by incorporating the ability to learn features from high-dimensional data such as multi-spectral images (Nagasubramanian et al. (2018); Simonyan and Zisserman (2014); Arad et al. (2018)), speech (Graves et al. (2013)) and text (Wang et al. (2012)). A particular form of deep networks called Convolutional Neural Networks (CNN) has become very popular due to its ability to autonomously create and analyze features in image-like inputs. Through the use of convolution operations, these models retain spatial neighbourhood information, thus allowing linking local (hierarchical) features of an image and an associated label, without the need for hand crafting of any features. Due to this special ability of ML algorithms to be input agnostic, i.e., the ability to automatically evaluate features from input data, they have found utility in a wide variety of applications including recommendation systems (Covington et al. (2016)) and self-driving cars (Bojarski et al. (2016)). These approaches are slowly gaining popularity in physics and engineered systems (Sladojevic et al. (2016); Stoecklein et al. (2017); Ghadai et al. (2018)), where modern sensor and computational developments have paved the
way for structured data generation (Sanchez-Lengeling and Aspuru-Guzik (2018); Dieb and Tsuda (2018)).

Here, we utilize the versatility of CNNs to map the active layer morphology of thin film OPVs to a performance metric, which is the short circuit current $J_{sc}$. Specifically, we train a morphology classifier that maps a OPV morphology to a short circuit current. We test several architectures (of varying depth and width) that can learn from a given set of morphologies and their labels, and demonstrate very high accuracy, and $F1$ score. To distinguish and rank order between these equally well performing models, we used two additional measures. The first is based on the observation that a good model must be able to generalize the learnt structure-property relationship. Thus, we identify network architectures that can generalize the map with the available data set. We quantify this in terms of the ability of the architecture to ‘project the unseen’ morphology onto the learnt distribution and make good predictions.

![Figure 4.1: DLSP (Deep Learning for Structure Property) framework: We construct a forward map from morphology to performance. Upon building trust in this trained model, we use it for performing manual exploration and insight buildings, as well as and automated design.](image)

Apart from generalizability, the other critical requirement for the ML model in our context is interpretability. While model interpretability is not a very critical metric for some applications (for
instance, network failure or stock pricing), it becomes a fairly important metric for understanding the behavior of engineered systems. This is because having a purely predictive ‘black-box’ model that is not interpretable raises a critical question: why should a domain expert believe in the prediction of a black-box model? This lack of “interpretability or explainability” is endemic to most black-box models and presents a major bottleneck to the widespread acceptance of ML models (Castelvecchi (2016)). Recently, there have been several approaches towards extracting interpretation from these “black-box” models (Castelvecchi (2016); Selvaraju et al. (2016); Ribeiro et al. (2016); Shrikumar et al. (2017a); Simonyan et al. (2013)). This includes domain-specific explanation of models (Ghosal et al. (2018b); Holzinger et al. (2017); Toda et al. (2019); Lee et al. (2019); Ghosal (2017); Ghosal et al. (2017, 2019)).

In the current context, the process of learning the structure-property relationship involves identifying several distinct local morphological traits (i.e. unsupervised feature learning) and weighing them appropriately to predict the performance of the morphology. While several (similarly performing) architectures will learn to look at multiple features, we argue that the most useful network is the one that can also identify the right features of the morphology used to make the (correct) prediction. In other words, the chosen architecture should be interpretable to gain trust in the model.

We introduce an approach called DLSP (Deep Learning for Structure Property interrogation) for learning the structure property relationship from data. Fig. 4.1 illustrates this approach graphically. We first construct a surrogate model of the structure-property relationship using a custom architecture based on a deep convolutional neural network. After training, this architecture is characterized for its trust using generalizability and interpretability measures. Specifically, generalizability is characterized by the performance of the models on off-sample morphologies, whose characteristics are not present in the training data set. Subsequently, interpretability is characterized by evaluating the “salient” features using saliency map visualizations. This dual characterization allowed us to pick a custom architecture over standard classifying architectures such as VGG-16 and ResNet50 architectures, all of which had nearly identical predictive power. We further use this trustworthy architecture to perform manual as well as automated explorations of the structure-property space.
Using a graphical web application we simplified the process of manual exploration and intuition building of the structure-property space. Here, the user can manually draw (2D) microstructures, perturb the microstructures and use the trained model to rapidly explore the impact of specific features on performance. Such analysis using a full scale physics model would require established, complex computing resources, which are generally not available to every researcher. Additionally, we integrated this trained model into an optimization framework to enable automated morphology. This work illustrates the substantial promise of such surrogate based design procedures in the design of complex multi-physics systems.

### 4.2 Results and discussion

#### 4.2.1 Training and Validation

We develop a CNN based architecture to classify morphologies into performance classes. A diverse set of binary morphologies were computationally created for use in training, testing and validation. We solved a thermodynamically consistent Cahn-Hilliard equation (Cahn (1961)) for binary phase separation using an in-house finite element library (Wodo and Ganapathysubramanian (2012b)). We ensured creation of a diverse set of morphologies by simulating systems with different volume fractions and different binary interaction parameters. As the Cahn-Hilliard equation models spinodal decomposition (or coarsening dynamics), we output morphologies at several time-snapshots for each simulation. A total of $\sim 65,000$ morphologies were generated. Each of these morphologies was computationally interrogated to evaluate the photovoltaic performance. The short-circuit current, $J_{sc}$, was evaluated for each morphology using the excitonic drift-diffusion equation (Kodali and Ganapathysubramanian (2012b)), which models photocurrent generation process in organic semiconducting films. Across the data set, the $J_{sc}$ exhibited a minimum of 0.6$mA/cm^2$ and a maximum of 7.0$mA/cm^2$. Subsequently, the continuous output, $J_{sc}$ was binned into 10 distinct equi-spaced bins, and each morphology was assigned a one-hot vector as its label.
(a) Proposed CNN architecture. Note that this is much shallower and has less trainable parameters compared to VGG-16 and ResNet-50.

(b) Confusion matrix for in-sample test predictions. Notice the heavily diagonally dominant matrix, indicating a very good classification accuracy.

**Figure 4.3:** Proposed architecture and in-sample testing results
4.2.1.1 Description of data:

The data set consists of images aggregated from solving the Cahn-Hilliard equations for a binary phase separating mixture with various blend ratios and interaction parameters (the complete data set is publicly available). Varying interaction parameters produce morphologies with different domain purities, while varying blend ratios produce domains of different sizes. Here, we choose to consider 2D morphologies, with extension to 3D morphologies being conceptually straightforward (but computationally non-trivial (Nagasubramanian et al. (2018); Ghadai et al. (2017)). This data set of morphologies (i.e. 2D, amorphous, isotropic) chosen is a sub-set of the diversity of morphologies that OPV films exhibit (amorphous-crystalline, anisotropic, and multi-phase). 1

We choose the short circuit current, \( J_{sc} \), as the output of the model. The performance of an OPV device is characterized by the current-voltage (JV) plot. The JV plot is completely parameterized in terms of three quantities, a) open circuit voltage \( V_{oc} \), b) short circuit current \( J_{sc} \) and c) fill factor. The \( J_{sc} \) explicitly depends on the morphology, while \( V_{oc} \) depends on the chemistry of the acceptor-donor materials (essentially the HOMO-LUMO gap). Consequently, this motivates our choice of \( J_{sc} \) as the output since it explicitly encodes the influence of morphology.

4.2.1.2 Neural network architecture:

Our custom network architecture for mapping a specific morphology to its label is depicted in Fig. 4.2a. It has 1.2 million learning parameters, consisting of 4 blocks comprised of a convolutional layer followed by a pooling layer (down sampling by \( 2 \times 2 \) max-pooling) followed by a batch normalization layer. The first and second blocks have 16 feature maps with \( 5 \times 5 \) convolutional kernels. The third block has 64 feature maps with \( 2 \times 2 \) kernels and the final block has 128 feature maps with \( 2 \times 2 \) kernels. After the final block, the output is flattened using a flatten layer and is followed by 3 fully connected (FC) layers with 512, 128 and 32 hidden units each, sequentially before reaching the final softmax output (prediction) layer of 10 units. A Dropout layer Srivastava et al. (2014b) with 50\% dropout was added between each of the FC layers. Training was performed on a

---

1Interestingly, we show our model trained on this strict subset of plausible morphologies performs well on morphologies representative of the larger OPV diversity, see Section 4.2.3.
total of 45,108 samples (with an additional 11,109 validation samples), and testing was performed on 11,109 samples. The learning rate was initiated at 0.0001. The Rectified Linear Unit ($ReLU$) function is used as the activation function for each of the convolutional and dense (FC) layers. To address over-fitting issues, we add dropout layers in between the fully-connected (FC) layers. The percentage of dropouts used was 50% after each of the fully-connected layers (namely, FC Layer 1, FC Layer 2 and FC Layer 3, as shown in Fig. 4.2b). After every convolutional and subsequent max-pooling layer, batch normalization was performed to remove internal covariate shift Ioffe and Szegedy (2015b). The network was trained for approximately 120 epochs (18s per epoch) with a learning rate of 0.0001, on the 45,000-image training set, giving an accuracy of 95.80%. The loss was denoted using a categorical cross-entropy function and Adam optimizer (Kingma and Ba (2014a)) was used to minimize the error.

Apart from this network, we also tested two standard architectures with our data set:

- **VGG-16** (learning parameters $\sim$ 50 million), with learning rate of 0.0001, batch size of 128 initialized with random weights was also trained on the training data set, achieving a test accuracy of 96.61% at epoch 70 (with 180s per epoch) with no further improvement in test accuracy.

- **ResNet-50** (learning parameters $\sim$ 23 million), with learning rate of 0.0001, batch size of 128 initialized with random weights was also trained on the training data set, achieving a test accuracy of 96.45% at epoch 70 (580s per epoch) with no further improvement in test accuracy.

A key point to note is that our network, although shallower, performs as well as the established deeper CNN models. Therefore, we select the network based on the learnt features (‘interpretability’) and out-of-sample performance (‘generalizability’) and not just the accuracy/f1-score of model on the testing data set. We also note that deeper networks also have additional problems – vanishing (or exploding) gradients Glorot and Bengio (2010), which hinder convergence, and the saturation of accuracy with increasing depth. We use saliency maps Simonyan et al. (2013) to visualize learnt features (Sec. 4.2.1.2), i.e., identify microstructure features used by the model to make classification
decisions. It is observed that the heat-maps signify the regions of varying degrees of importance and suggest a physical interpretation which is further discussed in Sec. 4.2.1.2.

4.2.2 Performance of models: Statistical metrics

A standard approach to quantify performance of a classification based machine learning framework is through the confusion matrix. Fig. 4.2b shows the confusion matrix for in-sample test data classification. It has an accuracy of 95.80% and F1-score of 97.28%. From the confusion matrix, it can clearly be seen that most of the classification is correct, and those which are incorrectly predicted are usually only off by one class. Some incorrect prediction is not unexpected, as we are binning a continuous variable into non-overlapping classes. As such, the edge cases have the potential to be misclassified. We also note that the other two standard architectures show similar confusion matrices, with similar prediction accuracy (see SI).

4.2.3 Out-of-sample testing to characterize model generalizability

It is a commonly known fact (Srivastava et al. (2014a)) that neural networks can possibly overfit, depending on the model capacity, amount of training data and training hyperparameters. The network thus memorizes the data and exhibits poor generalization capacity as well as brittleness (i.e. lack of robustness to perturbations). We, therefore, resort to two methods of checking the robustness of our trained network(s). As noted earlier, the morphology data used for training is generated by solving a PDE. This inherits certain properties to the data such as smooth contours and uniform domain sizes. Hence we try to systematically break these assumptions about the data set and see the performance of the network. First, we test the network on a columnar structure (Fig. 4.4). This structure is postulated as an ideal structure in literature (Kodali and Ganapathysubramanian (2013)). As the width of the columns decrease (and of the order of the exciton diffusion length) and the length of the columns increase, the performance of the morphology increases. This is an example of out-of-sample data – it has several sharp interface contours, which are completely absent in the training data set. The results of the performance of the models on this morphology are
Figure 4.4: Saliency maps and performance of our custom trained CNN. Note how the saliency maps closely follow the interface regions in the microstructure. It should also be noted that the networks shows good performance even on samples outside the training data set.

shown in Fig. 4.4. The actual $J_{sc}$ values from a full scale drift-diffusion simulation (along with the corresponding true label) are also presented. It is promising that the custom network accurately predicts the correct label corresponding to each of the columnar microstructures.

In a more difficult generalizability test, we use fractal like morphologies (Du et al. (2018)), that are constructed to maximize the interfacial area while minimizing the amount of tortuous transport. These ‘virtual’ morphologies have been shown to exhibit enhanced performance (Du et al. (2018)), but are currently difficult to experimentally fabricate. We make this point to emphasize that our training data set consists fully of morphologies that are experimentally feasible to fabricate. Our model correctly predicts the $J_{sc}$ class of all fractal like morphologies we considered (100% accuracy). It is very promising that our network has correctly identified (Fig. 4.4) all these as high performing class label 9. This provides substantial evidence of the generalizability of the model.
4.2.4 Building trust via interpretability characteristics

We next query the network to characterize the learnt features. We accomplish this using the concept of saliency maps (Simonyan et al. (2013); Montavon et al. (2017)) to identify the important features of the image input. Saliency mapping is a visualization technique that generates heat maps on images that bring out (highlight) the regions (microstructure regions, for our case) the trained CNN model focuses on to generate a classification output. Fig. 4.4 shows the saliency maps for morphologies in the data, columnar structures and the ”high” performing morphologies identified in Du et al. (2018).

We can see, in Figs. 4.4, how the network uses the interface between the acceptor and donor regions feature as a key measure for prediction. We believe this is critical evidence that makes this network trust-worthy. This is because the interface is the most critical feature affecting the performance. The length of the interface determines the amount of excitons that are dissociated. Additionally, interfaces that results in isolated islands or highly tortuous pathways result in enhanced recombination thus reducing performance. Finally, the impact of interfaces in the middle of the domain (away from the top and bottom electrodes) are more important, as the charges produced at these locations have a higher chance of recombination. We can see from Fig. 4.4 how the network is able to identify and utilize this interface information as critical to prediction of device performance.

Finally, we observe in Fig. 4.5 that the saliency maps from the standard deep networks (VGG-16 and ResNet-50) are unable to locate any interpretable features. Although the test accuracy of these networks is marginally higher than our custom network, we see that the saliency outputs from these networks do not provide us with any understandable information. Extensive numerical experimentation revealed that our model is shallow enough to provide meaningful saliency maps (i.e. be interpretable) while deep enough to produce accurate (and generalizable) predictions. We provide additional details in Appendix. 4.5.6. This observation is in line with Yosinski et al. (2015), where it was shown that deeper models are harder to explain than their shallower counterparts even though they may achieve a higher classification accuracy.  

These results signify the importance of tailoring architectures to the application. Thus, for performing morphology design, we use the custom architecture as a surrogate map from the microstructure space to the performance space.
Figure 4.5: Comparison of Saliency map outputs for our Custom Model (second column), VGG-net (third column) and ResNet-50 (fourth column), with input morphologies shown in the first column: top row shows an example image for class 0, bottom row shows an example image from fractal-like morphologies (correctly predicted as class 9 by our custom model)
4.3 Morphology design

Having developed a fast and trust-worthy surrogate map from microstructure to performance, we use it to enable microstructural design. In this section, we show two distinct applications, one manual and one automated, using this surrogate model for microstructures exploration and design. The goal of both these techniques is to explore and identify morphologies that demonstrate superior performance. Traditionally, this was generally achieved through a conventional optimization strategy, like simulated annealing, where an initial morphology is tweaked repeatedly to achieve superior performance. At every stage, the current morphology is evaluated for its performance. Subsequently, the whole process requires several computationally expensive evaluations and hence becomes time consuming. In the OPV context, evaluating the $J_{sc}$ for a 2D morphology requires access to dedicated high performance computing resources. While our highly optimized in-house excitonic-drift-diffusion (Kodali and Ganapathysubramanian (2012b,a)) code is able to perform one simulation in a few minutes on 24 processor, this is still not a viable approach for in-line design exploration and insight generation. In contrast, with the CNN based framework, evaluating the morphology becomes significantly faster and easier. Hence it provides an very powerful way to quickly ’evolve’ morphologies to reach morphologies with optimized performance.

4.3.1 Manual exploration and design insights

Using the surrogate, we created a browser (Fig. 4.6a) that enables the user to interactively modify morphologies to both visualize, test/build intuition and improve morphology performance. Using this interface, the user can get insights into the effect of morphological features on performance. Fig. 4.7 shows how one can modify morphologies to sequentially include several features of varying sizes, with the aim of improving performance. This tool can in turn help identify features of morphology that affect the performance. An example of this is demonstrated in Fig. 4.6b- 4.6j. It shows a set of morphologies along with the respective performance labels predicted by our network. First, we can see how performance can be improved from a simple bilayer by increasing the amount of surface area between the acceptor and donor (Wodo et al. (2015)). The maximum boost of
performance is obtained when the donor (black) domains are fractal like (Du et al. (2018)), as shown in Fig. 4.6e. Next, we add island type structures to inhibit performance (Wodo et al. (2015)). In our example, a ‘line’ of donor is added to the existing morphology, creating several acceptor domains unconnected to the cathode. The performance suffers drastically as informed by the physics of photoconversion (Kodali and Ganapathysubramanian (2012b)). This reduction can be compensated if the connectivities are improved for the acceptor, which can be seen in Fig. 4.6h. And finally, Fig. 4.6j shows how larger domains are not beneficial as they lead to geminate recombination and hence lower performance. Finally, a user can use approach as a design tool by incrementally adding changes to the initial morphology that can improve the predicted performance. Since the performance assessment is done by the trained CNN, the whole process happens in real-time.

4.3.2 Automated design

The above interface enables manual exploration and building of insight into the influence of various morphological features on performance. Manual exploration, however, inherently makes full exploration to find the best performing morphology manifold difficult and time-consuming. Thus, to fully explore this space, we link this fast surrogate with a probabilistic optimization algorithm to find promising, high-performing morphology classes. More specifically, we use a population based incremental learning (PBIL) approach to perturb morphologies and evolve them towards higher performance (Du et al. (2018)). PBIL estimates the explicit probability distribution of the optimal morphology. The multi variate probability distribution is stored as a probability matrix $P$ of the 2D morphology, i.e., each pixel is associated with a probability and is updated at each iteration to evolve towards promising morphology classes $^3$.

The integration of a robust and fast surrogate with a probabilistic exploration algorithm produces very promising results. Representative results are shown in Fig. 4.8 where the evolution of the

---

$^3$This matrix $P$ is updated as follows: the optimization starts with a given probability matrix, generally based on the intuition of the researcher. Subsequently, $n$ morphology instances are sampled around this matrix $P$. For each realization, the fast ML surrogate is deployed to evaluate the performance, $f_j, j \in [1,n]$. Then $n_b$ best samples ($n_b < n$) are used to calculate, $P_u$, the probabilistic update matrix. Next, the probability vector is updated according to $P = P \cdot (1 - l_r) + P_u \cdot l_r$, where $l_r$ is the learning rate. Intuitively, the update step reinforces features present in the best performing morphologies, and dampens those missing. The algorithm terminates by standard criteria (iteration limits and improvement bounds)
Figure 4.7: Manual exploration and insight building using the browser interface. Notice how several physics based intuitive trends can be identified and understood by incrementally perturbing the original bilayer morphology.
Figure 4.8: Exploration by semi-automated design: The optimization started with a bilayer structure. Notice how the framework directs the formation of finer features. Figure (d) shows the simulated electron and hold current densities under short circuit conditions for this optimized morphology. The result from automated design has been modified using physics based principles.

morphology is towards features with multiple scales, mimicking the finger-like fractal structures that are exhibited by high-performance morphologies (Du et al. (2018)). We perform full-physics simulations on one of the optimized morphologies (Fig. 4.8(c)), which confirms that the surrogate-derived morphology is in fact a high performing morphology (Fig. 4.8(d)).

4.4 Discussion

In this work, we address the computationally challenging issue of rapidly exploring morphology space to identify promising morphologies, especially in the context of multi-physics phenomena. While the approach is general, we illustrate the approach using the case of morphology tuning to enhance the performance of organic photovoltaic films. Our approach is a data-driven approach to learn a morphology quantifier that can perform fast evaluations. We train a custom designed CNN
maps a specified morphology into short circuit current, \( J_{sc} \), classes. Using out-of-sample data sets, we confirm absence of over-fitting issues during the training process. Two other standard networks (VGG-16 and ResNet-50) were also trained. It was observed that the custom network, although shallower, gave very similar accuracy. However, our custom network performed much better when visualized using saliency maps as well as when tested on out-of-sample data sets. It identified critical features of the interface in the morphology, which both VGG-16 and ResNet-50 failed to identify consistently. The custom designed network is then used to perform morphology design for achieving enhanced performance. Two approaches were taken to do this – the first one aims to inform the user about the effect of morphology on performance. The second approach uses the trust-worthy network as a fast cost function and performs morphology optimization using PBIL algorithm. This work serves as a proof of concept of using deep neural networks for material morphology quantification and design.

There are several interesting areas of research that this work suggests. First, we show that our model – though trained on a subset of plausible morphologies – is able to make accurate predictions on a much more diverse set of morphologies. This raises the question: 'What is the minimal diversity of morphologies that is needed for a trained model to be generalizable?' Such questions are particularly important to answer when data collection is resource intensive. Promising approaches include methods of active learning (Pokuri et al. (2018b)), and physics-aware models (Singh et al. (2018b); Shah et al. (2019)). Next, we show that CNN based surrogate models are promising approaches to rapidly explore structure-property manifolds. This raises the question: 'How can such techniques be extended to map and explore process-structure-property manifolds?' This question is particularly important to isolate promising processing windows that produce high performing devices. Promising approaches include surrogate models based on smart sampling (Pfeifer et al. (2018)), and ideas of manifold learning (Schoeneman et al. (2018)).
4.5 Methods

4.5.1 Organic Photovoltaics

Organic photovoltaic devices are energy harvesting devices which employ organic materials for solar energy conversion. These provide multiple advantages over traditional silicon based cells, like flexibility, transparency and ease of manufacturability. They however are limited by their efficiency of operation. Although major breakthroughs in processing and materials have improved the efficiency drastically, they still lag behind the traditional photovoltaics.

The efficiency of these devices is intricately dependant on the material distribution/morphology in the active layer. The active layer generally is a bulk hetero-junction, enabling multiple sites for energy conversion. Several features of the morphology have different roles in the process of converting solar energy. The ability to change these morphological features by changing the processing protocol is a major source of control in these devices.

The solar power conversion happens in several stages. Firstly, the incident solar energy generates excitons in the donor phase. These excitons are highly unstable and need to diffuse to a nearest interface with the acceptor material to separate into positive and negative charges. This diffusion to the interface is critical to evaluate the efficiency of absorption of incident light. These excitons dissociate at the acceptor-donor interface to form charges. The nature and quality of the interface has a direct impact on this efficiency. For example, interfaces with non-aligned crystal boundaries show lower dissociation than those with aligned crystals. In the next stage, these charges (positive charge in the donor and negative charge in the acceptor) are drifted to the respective electrode to produce electricity. Usually, this drift is provided by the potential difference between the two electrodes. However, these charges also encounter other interfaces which have pairs of positive and negative charges, leading to potential recombination.

In this context, quantifying the stage efficiencies (generation, dissociation and transport) becomes a critical part in developing strategies to design processing conditions. It can already be seen that the
role of morphology cannot be over-estimated in the power conversion efficiency. Hence strategies were
developed (Wodo et al. (2012); Marsh et al. (2010)) to quantify the efficiencies these morphologies.
While these techniques are robust and rigorous, they are expensive and time intensive. This makes
them infeasible for further designing morphologies, which often requires several quantifications.
So, we turn to modern fast methods of quantifying data, especially images. We represent the
morphologies as images and take advantage of deep convolutional neural networks to do performance
based classification.

### 4.5.2 Data generation and quantification

In order to train the network, we generate a data set of microstructures using a thermodynamic
consistent binary phase separation simulation. This is done by solving the well known Cahn-Hilliard
equation (Cahn (1961)) which tracks the local volume fraction of each material ($\phi_i$):

\[
\frac{\partial \phi_i}{\partial t} = \nabla (M(\phi_i) \nabla \mu_i)
\]

\[\mu_i = \frac{\partial f}{\partial \phi_i} - \epsilon^2 \nabla^2 \phi_i\]

$\frac{\partial \phi_i}{\partial t} = \nabla (M(\phi_i) \nabla \mu_i)$

$\mu_i = \frac{\partial f}{\partial \phi_i} - \epsilon^2 \nabla^2 \phi_i$  \hspace{1cm} (4.1)

$M(\phi_i)$ is the mobility of component $i$. $\mu_i$ represents the chemical potential of component $i$. The
chemical potential as defined in Eq.4.1 is the variational derivative of the total free energy of the
system. The total free energy comprises of the bulk free energy $f$ and the interfacial energy. The
interfacial free energy is characterized as $0.5\epsilon^2|\nabla \phi_i|^2$, where $\epsilon$ is the interfacial energy parameter.
$\epsilon$ is usually correlated with the thickness of the interface between the components. The bulk free
energy is described using the Flory-Huggins (Flory (1942)) energy representation:

\[f = \frac{\phi_1}{N_1} \ln \phi_1 + \frac{\phi_2}{N_2} \ln \phi_2 + \chi_{12} \phi_1 \phi_2\]

$\frac{\partial \phi_i}{\partial t} = \nabla (M(\phi_i) \nabla \mu_i)$

$\mu_i = \frac{\partial f}{\partial \phi_i} - \epsilon^2 \nabla^2 \phi_i$  \hspace{1cm} (4.1)

The degree of polymerization of the components is denoted by $N_i$ and $\chi_{ij}$ represents the severity
of interaction between the components. The values for $\chi$ are either estimated using molecular
simulations (Fu et al. (2013); Kok and Rudin (1982)), or experimentally (Orwoll (1977)), or calculated through empirical methods (Hansen (2007)).

This process generates time series of morphologies that can be treated as independent morphologies for the sake of training a machine learning model. This method helps to quickly produce several thousands of microstructures within a very short amount of time. In order to generate numerous consistent morphologies, we perform 100 simulations of the above equation (10 values of $\chi_{12}$ with 10 values of initial concentration), with morphologies outputted at every 20 timesteps (which provides distinguishable morphologies across timesteps). Previous analysis using this data can be found in (Wodo et al. (2015)). A characteristic of this procedure for generating morphologies through simulation is their similarity to morphologies in real active layers produced during thermal annealing, for example, the domains are similar in size and have smooth interface contours. These characteristics will also help us to build trust in the training process by manually creating morphologies that break these characteristics and testing the performance of the trained network on such samples. We produce a data set of nearly 65,000 (2D) gray-scale morphologies of size 101pix $\times$ 101pix.

These morphologies were then characterized using an in-house physics based simulator (Kodali and Ganapathysubramanian (2012b)). This simulator uses steady state excitonic drift diffusion equation to model the processes of exciton dissociation and charge transport:

$$J_n = -qn\mu_n \nabla \varphi + qV_t \mu_n \nabla n$$  \hspace{1cm} (4.3)

$$J_p = -qp\mu_p \nabla \varphi - qV_t \mu_p \nabla p$$  \hspace{1cm} (4.4)

$$\nabla . J_n = qfR_{[n,p]} - qfD_{[\nabla \varphi, X]}$$  \hspace{1cm} (4.5)

$$\nabla . J_p = qfR_{[n,p]} - qfD_{[\nabla \varphi, X]}$$  \hspace{1cm} (4.6)

$$\nabla . (\epsilon_r \epsilon_0 \nabla \varphi) = q(n - p)$$  \hspace{1cm} (4.7)

$$-\nabla . (V_t \mu_x \nabla X) - fD_{[\nabla \varphi, X]} - R_{[x]} = -G - R_{[n,p]}$$  \hspace{1cm} (4.8)

where $\mu_n, \mu_p$ are the mobilities of electrons and holes respectively. The quantities of interest are the electrostatic potential in the active layer $\varphi$, electron density $n$, hole density $p$ and exciton density.
X. \( G, D_{[\nabla \phi, X]} \) represent the rate of generation and dissociation of excitons respectively. \( R_{[x]} \) is the exciton relaxation rate. \( J_n, J_p \) are the current densities of electrons and holes respectively. We use the short circuit current \( J_{sc} \) as a means of labelling the data. The whole data was divided into 10 classes which are equally spaced between the best \( (J_{sc} = 7mA/cm^2) \) and worst performing \( (J_{sc} = 0.2mA/cm^2) \) in the data.

### 4.5.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have become the standard frameworks when it comes to computer vision tasks in recent times. To serve our purpose of classifying microstructures, we also use a CNN-based model to train on our data set, establish trust in the trained model and then use that trained model to make test/future predictions.

CNNs achieve a high level of performance with fewer parameters to learn (Krizhevsky et al. (2012); LeCun and Bengio (1998)) when compared to networks constructed simply via Fully-Connected (FC) layers. By design, they exploit the two-dimensional (2D) structure of an input image by preserving the locality of features and utilize spatially local correlations of an image by using tied weights, which are invariant to the translation of the feature positions (Krizhevsky et al. (2012); Lee et al. (2009)).

In CNNs, data are represented by multiple feature maps in each hidden layer. These feature maps are obtained by performing a local convolution of the input image using multiple filters. These feature maps further undergo non-linear down-sampling with a max-pooling operation (Boureau et al. (2010)) to decrease the data-dimension. Max-pooling partitions the input image into sets of non-overlapping rectangles and uses the maximum value for each partition as the output. This is done so that neighboring pixels in an image sharing similar features can be discarded. Both spatial and feature abstractness are also increased as a result, imparting increased position invariance for the filters (Boureau et al. (2010); Huang et al. (2007)).

We use batch normalization layers, which normalize the activations of the previous layer at each batch, to improving the overall performance of the architecture. Batch Normalization applies a
transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1 (Ioffe and Szegedy (2015b)).

Post max-pooling, multiple dimension-reduced vector representations of the input are acquired, and the process is repeated in the next layer to achieve a higher-level representation of the data. At the final pooling layer, the resultant outputs are linked to the FC layer, where Rectified Linear Unit (ReLU) activation outputs (Nair and Hinton (2010)) from the hidden units are joined to output units to infer a predicted class on the basis of the highest joint probability given the input data. Keeping this in mind, the probability of an input vector \( v \) being a member of the class \( i \) can be written as follows:

\[
Pr(Y = i|v, W, b) = \text{softmax}_i(Wv + b) = \frac{e^{W_i v + b_i}}{\sum_j e^{W_j v + b_j}}
\] (4.9)

where elements of \( W \) denote the weights and elements of \( b \) denote the biases. The model prediction is the class with the highest probability:

\[
y_{pred} = \text{argmax}_i Pr(Y = i|v, W, b)
\] (4.10)

The model weights, \( W \), and biases, \( b \), are optimized using error backpropagation algorithm (LeCun et al. (2015)), wherein true class labels are compared against the model prediction by using an error metric/loss function. We choose categorical cross entropy (Rubinstein (1999)) as the loss function, chosen to be minimized for the data set \( V \), and is given as follows:

\[
\mathcal{L}(V, Y) = -\frac{1}{n} \sum_{i=1}^{n} y^{(i)} \ln a(v^{(i)}) + (1 - y^{(i)}) \ln (1 - a(v^{(i)}))
\] (4.11)

Here, \( V = \{v^{(1)}, \ldots, v^{(n)}\} \) is the set of input examples in the training data set, and \( Y = \{y^{(1)}, \ldots, y^{(n)}\} \) is the corresponding set of labels for those input examples; \( a(v) \) represents the output of the neural network given an input \( v \).
4.5.4 Class specific visualization: Saliency Maps

A detailed description of Saliency maps and their use in visualising class specific regions as learnt by CNNs has been given in Simonyan et al. (2013). However, we here give a brief overview as well for the sake of simplicity. Saliency Map generation is a technique, which takes an input image, a learnt classification CNN model and a class of interest as it’s input and generates as an output, an image which is representative of that particular class in terms of what that learnt CNN model sees in the given input image. Formally, we define this as follows: Say, $\alpha_i(A)$ is the score of class $i$, computed by the classification layer of the CNN for an image $A$. The target is to find a $L_2$-regularized image such that $\alpha_i(A)$ is high:

$$\text{argmax}_A \alpha_i(A) - \gamma \|A\|_2^2$$

(4.12)

where $\gamma$ is the regularization parameter. Using the back-propagation algorithm (which is also used to optimize the layer weights), we obtain a locally optimal $A$ by optimizing with respect to the input image, with the model weights fixed to those obtained at the best-training step.

4.5.5 Performance of standard architectures

As discussed in Sec. 4.2.1, we tested the performance of our custom architecture with standard convolutional network architectures, namely ResNet-50 (He et al. (2015)) and VGG-16 (Simonyan and Zisserman (2014)). ResNet50 is a 50 layer deep convolutional network pretrained on images from ImageNet and can classify into 1000 object categories. It uses a special architecture called residual network blocks that simultaneously reduce the model size and capture diversity of input images. The final layer was modified to classify into 10 categories and was trained end-to-end with our data. VGG16 is another very popular architecture tested on data from ImageNet, which uses 13 layers of $3 \times 3$ convolutions with max pooling followed by 2 fully connected layers of 4096 neurons each. As with ResNet50, we modify the final layer of VGG16 to classify into only 10 categories. Although our architecture is shallower, it showed similar performance in terms of the confusion matrix. The confusion matrix on validation data for ResNet-50 and VGG-16 are in Fig. 4.10.
Figure 4.10: Both the standard architectures show performance similar to our custom architecture. But these do not provide any meaningful explanations to their predictions (Fig. 4.4)
4.5.6 How shallow can the network be?

In order to determine the simplest model with desired generalizable and interpretable characteristics, we performed an analysis of shallower variants of the presented architecture (Model $\alpha$). We trained a shallower model (Model $\alpha_{s1}$) retaining the first 3 convolution-max-pool-BN blocks of Model $\alpha$ (i.e., removing the last block from Model $\alpha$) as well as an even shallower model, $\alpha_{s2}$ which retains first two blocks of Model $\alpha$ (i.e., we remove the last 2 blocks from Model $\alpha$). The test accuracies reported for Model $\alpha_{s1}$ for the in-sample test data, fractal-like morphologies and the bilayer (refer Fig. 4.6b or Fig. 4.8 (a)) morphologies were 95.53%, 96.88%, 100% respectively and that for Model $\alpha_{s2}$ for in-sample test data, fractal morphologies and the bilayer morphologies were 95.12%, 56.65%, 50% respectively (Model $\alpha$ had classification accuracy of 96%, 100% and 98% for in-sample, fractal-like and bilayer morphologies).

Table 4.1 compiles the performance results of these models on three test data sets: in-sample morphologies, fractal-like morphologies, and bilayer morphologies. We observe that progressively shallower models perform worse in terms of prediction accuracy, especially for the out-of-sample data (fractal-like and bilayer morphologies). In other words, generalizability suffers when the models become shallower than the presented model (Model $\alpha$). This evidence suggests that Model $\alpha$ is the shallowest model that still produces viable accuracy.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy of in-sample morphologies</th>
<th>Accuracy of fractal-like morphologies</th>
<th>Accuracy of columnar morphologies (loss value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model $\alpha$</td>
<td>96%</td>
<td>100%</td>
<td>90% (4.22)</td>
</tr>
<tr>
<td>Model $\alpha_{s1}$</td>
<td>95%</td>
<td>96%</td>
<td>90% (14.56)</td>
</tr>
<tr>
<td>Model $\alpha_{s2}$</td>
<td>95%</td>
<td>57%</td>
<td>90% (18.34)</td>
</tr>
</tbody>
</table>
4.6 Data Availability

The data set, codes and the trained model used to generate the results are available through a Google Form request accessible through Github via the following link: Guided-Structure-Property-Exploration.

4.7 Conclusion

In this work, we address the issue of designing active layer morphologies to enhance device performance, especially in OPV applications. The usual methods to quantify morphology are either too costly or too simplistic. Hence we take a data-driven approach (DLSP) to create a morphology quantifier that can perform fast evaluations. We train a custom designed CNN that reads the morphology and classifies it into 10 bins of increasing performance metric $J_{sc}$. Using out-of-sample data sets, we confirm that there is no severe over-fitting issues during the training process. Two other standard networks (VGG-16 and ResNet-50) were trained end-to-end independently. It was observed that the custom network, although shallower, gave very similar accuracy. However, our custom network performed much better when visualized using saliency maps as well as when tested on out-of-sample data sets. It identified critical features of the interface in the morphology, which both VGG-16 and ResNet-50 failed to identify consistently. The custom designed network is then used to perform morphology design for achieving enhanced performance. Two approaches were taken to do this – the first one aims to inform the user about the effect of morphology on performance. The second approach uses the trust-worthy network as a fast cost function and performs morphology optimization using PBIL algorithm. It should be noted that this work serves as a proof of concept of using deep neural networks for material morphology quantification. It raises several other interesting questions of how to integrate physical phenomena into the training process. Can these physics based intuitions can be exploited to reduce the demand on the size of data for training? Can a most effective data set be created to reduce the training data? Can we make the training process more robust to adversarial attacks? All these questions form the scope of future study.
CHAPTER 5. REFERENCE BASED VISUALIZATION: TOP-K FEATURE MAPS, A SEMI-SUPERVISED FRAMEWORK FOR QUANTIFYING DISEASE SEVERITY IN SOYBEAN

In the last 2 chapters we explore HiL Deep Learning methods for label-efficient semi-supervised method generation for detection and counting of crops along with yield prediction in Chapter3 and also, how a trust based DL model can help classify and evaluate performances for complex microstructures used in semiconductor devices. In this Chapter, we introduce the concept of Reference-based HiLDL framework for another use-case, evaluating soybean plant diseases from leaf images where ICQP (Identification, Classification, Quantification and Prediction), a unique semi-supervised framework is proposed that utilizes domain knowledge and human verification during it’s testing phases.

Reference-based visualizations were proposed based on the concept of introducing scientific control to the visualizations. Upon generating the feature map using the input image, additional data that serve as a reference to the input image are also fed to the network for normalization. DeepLIFT (Shrikumar et al. (2017b)) utilizes vanilla backpropagation whose values are adjusted by the gradient of an additional reference image. Explanation map, which is a hybrid approach of DeepLIFT and Grad-CAM (Selvaraju et al. (2016)), can handle batches of reference images for normalization by first calculating the mean activation value of the respective neurons when the reference images are fed and then defining an activation threshold for normalization. Instead of using the gradients, the activation thresholds are used for normalizing the intermediate outputs and the sum of the top three highly activated outputs are used for attention map generation. We apply the top-k feature map framework to the first (i.e., shallowest) convolution layer with healthy leaf images as a reference that can specifically highlight the disease lesions within the soybean leaf image.
Figure 5.1: Schematic illustration of foliar plant stresses in soybean grouped into two major categories, biotic (bacterial and fungal) and abiotic (nutrient deficiency and chemical injury) stress. The images were used to develop the DCNN for the following eight stresses: bacterial blight (*Pseudomonas savastanoi* pv. *glycinea*), bacterial pustule (*Xanthomonas axonopodis* pv. *glycines*), sudden death syndrome (SDS, *Fusarium virguliforme*), Septoria brown spot (*Septoria glycines*), frogeye leaf spot (*Cercospora sojina*), IDC, potassium deficiency and herbicide injury. For each stress, information such as symptom descriptors, areas of appearance and most commonly mistaken stresses that exhibit similar symptoms are listed. These particular foliar stresses were chosen because of their prevalence and confounding symptoms.

### 5.1 Introduction

Conventional plant stress identification and classification has invariably relied on human experts identifying visual symptoms as a means of categorization (Bock et al. (2010)). This process is admittedly subjective and error-prone. Computer vision and machine learning have the capability of resolving this issue and enable accurate, scalable high-throughput phenotyping. Among machine learning approaches, deep learning has emerged as one of the most effective techniques in various fields of modern science such as medical imaging applications that have achieved dermatologist level classification accuracies for skin cancer (Esteva et al. (2017)), in modeling neural responses and population in visual cortical areas of the brain (Yamins and DiCarlo (2016)) and in predicting sequence specificities of DNA- and RNA-binding proteins (Alipanahi et al. (2015)). Similarly, deep learning based techniques have made transformative demonstrations of performing complex cognitive
tasks such as achieving human level or better accuracy for playing Atari games (Mnih et al. (2015)) and even beating a human expert in the game Go (Silver et al. (2016)).

Here, we build a deep learning model that is exceptionally accurate in identifying a large class of soybean stresses from RGB images of soybean leaves (see Fig. 5.1). However, this type of model typically operates as a black-box predictor and requires a leap of faith to believe its predictions. In contrast, visual symptom-based manual identification provides an explanation mechanism (e.g., visible chlorosis and necrosis are symptomatic of iron deficiency chlorosis [IDC]) for stress identification. The lack of “explainability is endemic to most black-box models and presents a major bottleneck to their widespread acceptance (Castelvecchi (2016)). Here, we sought to look under the hood of the trained model to explain each identification and classification decision made. We do so by extracting the visual cues or features responsible for a particular decision. These features are the top-K high-resolution feature maps learnt by the model based on their localized activation levels. These features, which are learnt in an unsupervised manner, are then compared and correlated with human-identified symptoms of each stress, thus providing an inside look at how the model makes its predictions.

5.2 Materials and Methods

5.2.1 Leaf Sample Collection and Data Generation

We started with the collection of images of stressed and healthy soybean leaflets in the field. The labeled data was collected following a rigorous imaging protocol using a standard camera. The imaging platform 5.2 comprised a flat rectangular surface covered with a dark cloth used to limit background noise. The platform was shaded by a large umbrella to ensure a consistent light source during imaging. Four leaves at a time were placed along the four corners of the platform, a color chart was placed in the center, and the leaves and chart were manually imaged using a digital camera (Canon EOS Rebel T5i, 18 mega-pixels). Using image processing techniques, the original images were segmented into four separate leaf images. The images were appropriately labeled as one of the eight different stress classes on the basis of the earlier field diagnoses. Each soybean stress
class contained approximately 2000 leaf images with varying degrees of severity, whereas the healthy soybean class contained over 5000 images. Over 25,000 labeled images (data set available online here) were collected to create a balanced data set of leaflet images from healthy soybean plants and plants exhibiting eight different stresses (Fig. 5.1). Leaflet images were taken from plants in soybean fields across the state of Iowa, in the United States. This data set represents a diverse array of symptoms across biotic (e.g., fungal and bacterial diseases) and abiotic (e.g., nutrient deficiency and chemical injury) stresses.

Figure 5.2: Leaf Imaging Platform

5.2.2 Data Collection

A total of eight different soybean stresses were selected for inclusion in the data set, on the basis of their foliar expression and prevalence in the state of Iowa. The eight soybean stresses included the following: bacterial blight, bacterial pustule, Septoria brown spot, sudden death syndrome
Figure 5.3: Example leaf images of a healthy leaf and each stress class

Figure 5.4: Illustration of data augmentation scheme

(SDS), frog-eye leaf spot, herbicide injury, potassium deficiency, and iron deficiency chlorosis (IDC) (Koenning and Wrather (2010)). Healthy soybean leaflets were also collected to ensure that the machine learning model can successfully differentiate between healthy and stressed leaves. First, various soybean fields in central Iowa associated with Iowa State University were scouted for the desired plant stresses. Entire plant samples were collected directly from the fields and taken to the Plant and Insect Diagnostic Clinic at Iowa State for official diagnosis by expert plant pathologists; for more information and online access, please follow this link to the online Plant and Insect Diagnostic Clinic. The exact locations of the sampled soybean plants were recorded at that time. After the stress identities were confirmed by the Plant and Insect Diagnostic Clinic, the desired fields were
revisited. Individual soybean leaflets exposed to a range of severity levels were then identified and collected manually through destructive sampling. Stresses such as frog-eye leaf spot, potassium deficiency, bacterial pustule and bacterial blight were present at low to medium severity. The leaflets were placed into designated bags and taken to an on-site imaging platform.

5.2.3 Data Preparation and Generation

The data set for training, validation and testing was prepared in the following manner: First, the images of the leaves were segmented out from the raw images and reshaped into images of pixel size $64 \times 64$ [(height) $\times$ (width)] for efficient training of the deep neural network. We used 4174 images for healthy leaves, 1511 images for bacterial blight, 1237 images for brown spot, 1096 images for frogeye spot, 1311 images for herbicide injury, 1834 images for IDC, 2182 images for potassium deficiency, 1634 images for bacterial pustule and 1228 images for SDS, i.e., a total of 16207 clean images. Refer to Fig.5.3 showing example images for each class.

A standard data augmentation scheme was adopted to enhance the size of the data set. We augmented 1096 images for each of the stress classes and augmented 2192 images from the healthy class. The following augmentations were conducted: horizontal flip, vertical flip, 90 degree clockwise (CW) rotation, 180 degree CW rotation and 270 degree CW rotation. (see Fig. 5.4 for an illustration of the data augmentation scheme) The total data set consisted of 65760 images, which were then divided into training, validation and test sets in a 7:2:1 proportion. Each image in this data set is associated with an expert marked label indicating a stress class. For a small subset of the images ($\sim 1000$ images), we collected details of the visual symptoms that the expert pathologists used to identify a particular stress. This information was only used to quantify the explainability of the framework and was not used anywhere in the training process.

5.3 Deep CNN Model and Explanation Framework

We built a deep convolutional neural network- (DCNN-) based supervised classification framework (the model is available here to identify and classify stresses (Fig. 5.5 (a) and (b)). DCNNs have
Figure 5.5: (a) DCNN architecture used. (b) Explanation Phase. The concept of isolating the top-K high-resolution feature maps learnt by the model based on their localized activation levels was applied to automatically visualize important image features used by the DCNN model shown an extraordinary ability (Esteva et al. (2017); Yamins and DiCarlo (2016); Alipanahi et al. (2015); Mnih et al. (2015); Silver et al. (2016); Sladojevic et al. (2016); Ubbens and Stavness (2017)) to efficiently extract complex features from images and function as a classification technique when provided with sufficient data. The exhibited accuracy is especially promising, given the multiplicity of similar and confounding symptoms between the stresses in single crop species (see Fig. 5.1). We associate this classification ability with the hierarchical nature of this model, which is able to learn features of features from data without the time-consuming hand-crafting of features. During deployment of the model for classification inference on a leaf image, we isolate the top-K high-resolution feature maps based on their localized activation levels. These feature maps indicate which regions in the leaf the DCNN model uses to perform the classification.

5.3.1 Network Parameters

After an exhaustive exploration of various applicable DCNN architectures and their classification and explanation capabilities, we choose the network shown in Fig. 5.5. This DCNN architecture
consists of 5 convolutional layers (128 feature maps of size 3 × 3 for each layer), 4 pooling layers (down sampling by 2 × 2 max-pooling), 4 batch normalization layers and 2 fully connected (FC) layers with 500 and 100 hidden units each, sequentially. 3 Dropout layers were added. 2 of these (with a dropout rate of 50% for each) were added after each of the fully connected (FC) layers and another after the Flatten layer (with the same dropout rate). The learning rate was initialized with 0.001. Training was performed using a total of 53,265 samples (with an additional 5,919 validation samples), and testing was performed on 6,576 samples. The first convolutional layer maps the 3 channels (RGB) in the input image to 128 feature maps by using a 3 × 3 kernel function. Subsequent max-pooling decreases the dimensions of the image. Max-pooling is performed by taking the maximum value in the kernel window that is passed over the image. The stride here is the default stride, i.e., 2, which means that the window is moved 2 pixels at a time. This decrease in image dimension not only picks up key features, but also reduces computation complexity and time (Dumoulin and Visin (2016)). The Rectified Linear Unit (ReLU) function is used as the activation function, because of significant advantages over other activation function choices. ReLU increases the speed of training and requires very simple gradient computation. In the context of deep neural networks, the rectifier is an activation function, defined as: \( f(x) = \max(0, x) \), where \( x \) is the input to a neuron. A unit utilizing the rectifier is referred to as the ReLU. We trained the model using a NVIDIA GeForce GTX TITAN X (12 GB memory) with CUDA 8.0 (and cuDNN 5.1). See the next section for a detailed discussion of training.

### 5.3.2 Training the DCNN

Our designed architecture initially had close to 3 million learning parameters, which decreased to 901,877 (with 901,653 trainable and 224 non-trainable parameters) while the same level of prediction accuracy was maintained. Although deep neural networks with a large number of learning parameters are very powerful architectures, overfitting becomes a severe problem in these cases. Large networks such as these are also slow to train, thus making it more difficult to address overfitting issues by combining the predictions of many large neural networks during the testing phase. Adding Dropout
layers can usually solve such overfitting issues. The idea behind adding dropout layers is to randomly drop units along with their connections from the neural network during training (Srivastava et al. (2014a). The percentage of dropouts used in our model is depicted in Fig. 5.5. The last FC layer after the final dropout layer gives the prediction for the class to which the input image belongs. After every convolutional layer, batch normalization was performed to remove internal covariate shift (Ioffe and Szegedy (2015a)). The network was trained for approximately 100 epochs on the 53,265-image training set to reach the desired accuracy. The cross-entropy (categorical) loss (or cost) function along with Adam optimizer (Kingma and Ba (2014b)) was used to minimize the error. Adam was chosen as the optimizer primarily because it requires minimal tuning of its internal hyperparameters. All other hyper parameters mentioned above have been cross-validated and chosen on the basis of repeated experiments to achieve the best possible prediction accuracy and explanation. Note that it is a common observation made by the research community that certain compromise may be required in terms of accuracy to obtain a more explainable model. Our hyperparameter exploration involved varying the initial learning rate (e.g., 0.001 to 0.0001), mini-batch size (20 to 100), extent of zero padding (no zero padding to 2 zero padding). The prediction accuracy varies from 92.71% to 95.04% over all the hyperparameter variations mentioned here. Our main observation was that while with larger batch size and padding, the accuracy improves, but the explanation quality may deteriorate. Based on this trade-off between accuracy and explanation quality, we have chosen a a model that achieved 94.13% accuracy as well as excellent explanation quality. A common practice in the community is to transfer standard models pre-trained with benchmark data sets to a different domain data with limited fine-tuning. For example, in our case, we tested the VGG-16 architecture pre-trained with the Imagenet data set. While we were able to achieve up to 90.92% classification accuracy, the explanation performance using the proposed technique was quite poor. Hence, we preferred our custom model to demonstrate the efficacy of the xPLNet framework. Fig. 5.13 shows how the prediction accuracy varies with the availability of training data.
5.3.3 DCNN explanation framework for severity classification and quantification

We develop a DCNN explanation approach focused on identifying the visual cues (i.e., stress symptoms) that are used by the DCNN to make predictions. The availability of these visual cues used by the model increases our confidence in the model predictions. In this context, we observe that our DCNN model captures color-based features at the low abstraction levels (i.e., first or second convolution layer) which conforms to the general observation of deep neural networks capturing simple low-complexity, but important explainable features at the lower layers (Shrikumar et al. (2017c)). Similarly, visual stress symptoms that a human perceives can also be described using color-based features. Therefore, these feature maps from the model can serve as indicators of visual stress symptoms as well as a means of quantifying the stress severity. We provide a brief description of our explanation approach below, with detailed mathematical formulations and algorithms available below.

We begin with picking a low-level convolution layer in the model for isolating feature maps with localized activation levels. We compute the probability distribution of mean activation levels of all the feature maps for all the healthy leaf images\(^1\). Assuming this distribution to be Gaussian, we pick a threshold (mean $+3\sigma$), called as the stress activation (SA) threshold, on the mean activation levels beyond which the activation levels are considered to be indicators of a stress (similar to the notion of a reference image in explanation techniques such as DeepLIFT (Shrikumar et al. (2017a))).

During inference with an arbitrary leaf image, we rank-order the feature maps at the chosen layer based on Feature Importance (FI) metric. This metric is based on the mean activation level of each feature map, computed over those pixels with activation levels above the SA threshold computed earlier. We then consider the $K$ top ranking feature maps. We observed that most of the activation energy is captured by the top 2 or 3 feature maps among the 128 feature maps generated at the first convolution layer. We use $K = 3$ in the rest of our results. An explanation map (EM) is then generated by computing a weighted average of the top-K feature maps with the FI metrics as their weights. We find that this EM is highly correlated with the visual cues used by an expert rater to

\(^1\)The mean activation levels are computed for the foreground only (i.e., for the leaf area), to ensure that no background information is used
identify symptoms, and quantify stress severity. Therefore, the mean intensity of the explanation map serves as a percentage severity level (0% being a leaf with no symptoms and a high value indicating significant symptoms) which can be discretized to provide a stress severity class. We consider a standard discretized severity scale - 0-25%: resistant, 25-50%: moderately resistant, 50-75%: susceptible, and 75-100%: highly susceptible.

5.3.3.1 Isolating top-K feature maps based on localized activation levels

While the DCNN has been trained with only class-labeled images, we develop a DCNN explanation approach focused on identifying the pixel-level visual cues (i.e., stress symptoms) that are used by the DCNN to make predictions. Such an explanation framework enables us to estimate the severity level of the identified stress in a completely unsupervised manner. Recently, there has been a significant recent interest in the deep learning community to develop such explanation methods. However, the community is still far from a consensus on a most effective technique as most of the methods have their own pros and cons and may not be useful for every problem at hand. For example, among various model agnostic explanation techniques, Local Interpretable Model-Agnostic Explanations (LIME) (Ribeiro et al. (2016)) has been shown to be useful for certain problems. However, it involves a stiff optimization problem that may become intractable for many test image examples. Among the gradient based approaches, Grad-CAM (and its variant guided Grad-CAM) (Selvaraju et al. (2016)) has been successful in isolating class discriminative features in the input image. Still, it does have several issues related to vanishing gradients, low resolution of the explanation and it may not be quite transparent in understanding what features the network has learned. Hence, it needs complex post-processing. Therefore, we have developed a simpler approach specific to our problem at hand that is a more robust (i.e., works for most test images), high resolution and transparent method based on the concept of isolating top-K high-resolution feature maps learnt by the model based on their localized activation levels.

Note that while the explanation map will be already at a similar resolution level as the input image due to the choice of a lower abstraction layer (before any downsampling/pooling occurs), it can be extrapolated to the exact same resolution as the input image if required.
We begin with picking a low-level convolution layer in the model (that primarily captures the color-based features for the leaf images) for isolating feature maps with localized activation levels. Upon choosing a layer, the first task is to find a reference activation level in the feature maps at that layer. Note that the concept of a reference activation level is motivated by the recently proposed DeepLIFT approach (Shrikumar et al. (2017a)) that poses the question of feature importance in terms of difference from an appropriately chosen ‘reference state’. This approach has been shown to resolve the vanishing gradient and gradient discontinuity issues faced by gradient based methods such as Grad-CAM. However, choice of a reference image or activation may not be quite straightforward. In our problem though, feature maps from healthy leaf images presents a natural reference point,
Figure 5.7: Illustration of the concept of isolating top-K high-resolution feature maps learnt by the model based on their localized activation levels

difference from which should indicate the stress features. Therefore, we compute the probability distribution of mean activation levels (over all foreground/leaf pixels) of all the feature maps using the all the healthy leaf samples. The distribution is shown in Fig. 5.6 which can be assumed to be a Gaussian distribution. As a reference for activation levels, a threshold (mean $+3\sigma$) on the mean activation levels is chosen as the Stress Activation (SA) threshold $A_{SA}$.

Upon computing this reference, $A_{SA}$, let us consider a test leaf image. Let $A(u,v)$ be the activation levels of a feature map in the pre-selected layer corresponding to the test image, where $(u,v)$ is the pixel location within the feature map (foreground only). We define the pixel-level importance quantity, $\Delta A(u,v)$ as

$$\Delta A(u,v) = A(u,v) - A_{SA} \quad (5.1)$$
We also define an indicator function $I(u,v)$ as

$$I(u,v) = \begin{cases} 
1 & \text{if } \Delta A(u,v) > 0 \\
0 & \text{if } \Delta A(u,v) \leq 0
\end{cases}$$

With these notations, we are now ready to define a Feature Importance (FI) metric, $\Delta A$ for a feature map as

$$\Delta A = \frac{\sum_{(u,v)} I(u,v) \odot \Delta A(u,v)}{\sum_{(u,v)} I(u,v)}$$

where, $\odot$ denotes element-wise multiplication. We rank-order the feature maps based on the feature map importance, $\Delta A$ to select the $K$ top ranking feature maps. Fig. 5.8 shows examples of such ranking and we observe that most of activation energy is captured by the top 2 or 3 feature maps among the 128 feature maps generated at the first convolution layer and hence, we use $K = 3$ for our results. Now, an explanation map, $EM(u,v)$ for a test image is computed as

$$EM(u,v) = \sum_{i=1}^{K} \Delta A_i \cdot I_i(u,v) \odot \Delta A_i(u,v)$$
where, subscript $l$ denotes the index for the top-K feature maps for all the quantities involved. We use this explanation map as a means to quantify stress severity. The mean intensity of the explanation map (in grayscale) serves as a percentage severity level which can be discretized to provide a stress severity class.

We summarize the explanation technique in an algorithmic format below and provide a succinct illustration in Fig. 5.7.

**Algorithm 1** Top-K Feature Map explanation

1. **Reference Generation:** Input: Trained DCNN model, Choice of explanation layer, $X_{train}^H$ (only healthy leaf samples)
2. Compute mean activation levels (over all foreground/leaf pixels) of all the feature maps (in the chosen explanation layer) using all healthy leaf samples.
3. Compute sample mean and sample $\sigma$ for the mean activation level population.
4. Compute Stress Activation (SA) threshold $A_{SA}$ as mean +3$\sigma$.
5. **Explanation Map Generation:** Input: Trained DCNN model, Choice of explanation layer, $K$, $A_{SA}$, $X_{test}$
6. Identity activation levels, $A(u, v)$ for all foreground pixels (pixel location - (u,v)) of all Feature Maps in the chosen explanation layer corresponding to the test leaf image denoted by $FM$.
7. Compute pixel-level importance, $\Delta A(u, v)$ based on Eq. 5.1 $\forall FM$.
8. Compute indicator function, $I(u, v)$ based on Eq. 5 $\forall FM$.
9. Compute Feature Importance (FI) metric, $\Delta A$ based on Eq. 5.1 $\forall FM$.
10. Rank $FM$s in a descending order based on the FI metric and isolate top-K $FM$s.
11. Compute Explanation Map, $EM(u, v)$ based on Eq. 5.2.
12. Compute mean intensity of $EM(u, v)$ (in grayscale) $\rightarrow$ Plant Stress Severity Quantification.
13. Discretize mean intensity of $EM(u, v)$ (in grayscale) $\rightarrow$ Plant Stress Severity Classification.

We schematically summarize the explainable deep learning framework used for plant stress identification, classification and quantification in the Fig. 5.5 and also provide an algorithmic summary of the overall xPlNet framework in Algorithm 2.

### 5.4 Results

#### 5.4.1 Stress Identification

In this section we present the results based on a DCNN model built after hyperparameter exploration that provides a good balance between high classification accuracy and explainable visual
Algorithm 2 xPlNet

1: **Training Phase:** Input: $X_{train}, Y_{train}$: $x$: RGB leaf images, $y$: class labels
2: Select DCNN architecture and hyper-parameters.
3: Learn DCNN model parameters using $X_{train}, Y_{train}$.
4: **Testing Phase:** Input: Trained DCNN model, $X_{test}$
5: Compute DCNN inference for test data $\rightarrow$ Plant Stress Identification.
6: **Explanation phase:** Input: Trained DCNN model, Choice of explanation layer, $X_{train}$ (only healthy leaf samples), $X_{test}$
7: Generate Stress Activation threshold for the chosen explanation layer based on the healthy leaf training images.
8: Isolate top-K feature maps from the chosen explanation layer for the test sample using on a feature importance metric based on localized activation levels beyond the reference Stress Activation threshold.
9: Generate Explanation Map via weighted averaging the top-K feature maps using the feature importance metric as weights.
10: Compute mean intensity of the explanation map (in grayscale) $\rightarrow$ Plant Stress Severity Quantification.
11: Discretize mean intensity of the explanation map $\rightarrow$ Plant Stress Severity Classification.

symptoms. We emphasize that the best performing model in terms of classification accuracy need not necessarily provide the best explanation. In Fig. 5.9, we present qualitative results of deploying the trained DCNN for stress identification, classification and quantification, while quantitative results over the full test data set are reported in Fig. 5.10 and Fig. 5.11. We found a high overall classification accuracy (94.13%) using a large and diverse data set of unseen test examples (approximately 6000 images, with around 600 examples per foliar stress). The confusion matrix revealed that erroneous predictions were predominantly due to confounding stress symptoms that cause confusion even for expert raters (Fig. 5.10). For example, the highest confusion (17.6% of bacterial pustule test images predicted as bacterial blight and 11.6% of bacterial blight test images predicted as bacterial pustule) occurred between bacterial blight and bacterial pustule; discriminating between these two diseases is challenging even for expert plant pathologists due to confounding symptoms (Hartman et al. (2015)).
5.4.2 Symptom Explanation and Severity Quantification

We compare the machine-based explanation maps with human expert ratings (which can suffer significantly from inter-rater variability, see Discussions section), by evaluating the spatial correlation function between the expert marked visual cues and the machine explanation map. Using the spatial correlation function not only compares total intensity, but also the spatial localization of the visual cues. Fig. 5.11 shows the comparison using spatial correlations between the two sets of ratings (both represented in grayscale) for four different stresses (for which we had sufficient number of representative samples across various stress levels), namely, IDC, SDS, Septoria Brown spot and Herbicide injury. Results show a high level of agreement between the machine and human ratings which proves the viability of a completely unsupervised severity quantification technique based on an explainable DCNN framework. This allows us to avoid the very expensive pixel-level visual symptom annotations. We remind the reader that we had symptom annotations only to validate the machine-based severity ratings. We show a representative set of examples for each stress along with their explanation maps and human annotations in Fig. 5.9. The close similarity between the expert annotation with the explanation map significantly increases our confidence in the predictive
capability of the model. We subsequently use the explanation maps to compute severity percentages for these examples based on mean intensities of the explanation maps; and compute severity classes by discretizing the severity percentages as described earlier. Please see Discussions section for more representative examples.

5.4.3 High-throughput Deployment and Transfer Learning Capability

Well-trained DCNNs learn to generalize features rather than memorize patterns (LeCun et al. (2015)). We explore this characteristic and test whether the DCNN trained with a specific imaging protocol and targeted for soybean stresses could make accurate predictions under other imaging conditions and for other plant species with the same stresses. This capability for transfer learning (Mohanty et al. (2016)) was investigated with several test images with IDC, Potassium deficiency and SDS symptoms using non-destructive imaging protocols (e.g., canopy imaging with hand-held camera). With 62 such test examples, we obtain a stress identification accuracy of 90.32% (see details about the data collection strategies and test examples in the Discussions section). Such performance of the model under different illumination conditions demonstrates the possibility of deploying this framework for high-throughput phenotyping. We also show anecdotal success for a few non-soybean leaf image examples (e.g., IDC in cucurbits and Potassium deficiency in oilseed rape) with reasonable quality from the internet (see Discussions section). While such results are very promising, we refrain from drawing any firm conclusions due to the lack of availability of a statistically significant data set of non-soybean leaf images with stress symptoms.

5.5 Discussions

The identification of human-interpretable visual cues provides users with a formal mechanism ensuring that predictions are useful (i.e., determining whether the visual cues are meaningful). Additionally, the availability of the visual cues allows for the identification of stress types and severity classes that are under-performing (those in which the visual cues do not match the expert-determined

\[ \text{We observed that the very few deviations in these results were primarily due to the low quality of the input images, which exhibited shadows, low resolution and a lack of focus, see Discussions} \]
symptoms), thus potentially leading to more efficient retraining and targeted data collection. Here, we emphasize that the identification of visual symptoms involves a completely unsupervised process that does not require any detailed rules (e.g., involving colors, sizes and shapes) to identify the symptomatic regions on a leaf; hence, this process is extremely scalable. Furthermore, the automated identification of visual cues could be used by plant pathologists to identify early symptoms of stress. In the context of plant stress phenotyping, four stages of the problem are defined (Singh et al. (2016)), namely, identification, classification, quantification and prediction (ICQP). In this work, we provide a deep machine vision-based solution to the first three stages. The approach presented here is widely applicable to digital agriculture and allows for more precise and timely phenotyping of stresses in real time. We show that this approach is reasonably robust to illumination changes, thus providing a straightforward approach to high throughput phenotyping. Similar models can be trained with data from a variety of imaging platforms and on-field protocols (UAV’s, ground imaging, satellite), and various growth stages. We envision that this approach could be easily extended beyond plant stresses (i.e., to animal and human diseases) and other imaging modalities (hyper-spectral) and scales (ground and air), thereby leading to more sustainable agriculture, food production, and health-care.

### 5.5.0.1 DCNN performance for Non-destructive Images and Non-soybean Leaves

We have also tested the DCNN framework on non-destructive imaging of leaflets of stressed soybean canopies to illustrate its ability to classify stresses in high-throughput settings as well. A standardized imaging protocol was used to collect the non-destructive leaflet images for different stresses, including iron deficiency chlorosis (IDC), Potassium deficiency and sudden death syndrome (SDS). First, in a similar fashion to the destructive leaflet imaging, different soybean fields were scouted for various stresses of interest. Next, selected plant samples were sent to the Plant and Insect Diagnostic Lab at Iowa State University for official stress identification. Once stress identities were confirmed, the field locations were revisited, and appropriate soybean plants were selected for imaging. Using a digital camera (Canon EOS Rebel T5i 18 mega-pixels), canopy images with
stresses of interest were collected. Individual leaflets exhibiting the stresses were segmented out from the canopy into new separate images through image processing techniques and grouped into their respective stress classes based on the field diagnosis. Examples of this protocol is shown on the middle and the right side of the top row of Fig. 5.15.

A standardized non-destructive imaging protocol utilizing a clipboard imaging platform was deployed to acquire non-destructive leaflet images for IDC. The clipboard imaging platform was fitted with blue paper and a mini color chart. A shadow was cast over the soybean plant of interest to guarantee consistent lighting. Next, one leaflet at a time was scooped onto the surface of the clipboard while still attached to the plant itself. Special emphasis was made to ensure the leaflet was lying completely flat on the clipboard surface. Using the camera of an Apple iPhone 6s smartphone, single non-destructive leaflet images for the stress of interest were collected. Through image processing techniques, each leaflet was segmented out from its original image into a separate new image for analysis purposes. Examples of this protocol is shown on the left side of the top row of Fig. 5.15.

We evaluated the DCNN model with several leaflet images exhibiting IDC, K-deficiency and SDS symptoms. The DCNN based stress identification process accurately identified 20 out of 22 IDC samples, 21 out of 22 Potassium deficiency samples, 15 out of 18 SDS samples. This corresponds to an overall accuracy of 90.91% using a total of 62 samples. Fig. 5.15 shows a few representative examples along with their symptom explanations by the xPlNet framework.

There are various visual differences between destructive and non-destructive leaflet samples due to the nature of their respective image collection protocols. The most important difference is that non-destructive leaflets are not manipulated to lie flat on an imaging platform before image collection. These leaflets are still attached to its soybean plant. Thus more curvature is present, which creates additional situations for glare and shadowing that can affect the DCNNs classification decisions. On the other hand, non-destructive leaflets do not experience instances of wrinkling due to moisture loss, while this is a typical experience for destructive leaflets in the summer heat. The
drying out and wrinkling of those leaflets would reduce the quality of those leaflet images. We notice that even under this conditions, the DCNN framework is able to perform reasonably accurately.

Another aspect of transfer learning is to leverage the current model trained on soybean leaves for non-soybean leaves and we present a few anecdotal examples here with images collected from the internet for IDC in cucurbits and potassium deficiency in oilseed rape. The examples shown in Fig. 5.16 demonstrates that the model is able to accurately identify the stresses and their symptoms in non-soybean plants.

5.5.1 Severity Rating by Human Experts using a Visual Application

Multiple expert raters were also given a subset of images from our test set to rate and annotate the symptoms in the leaf images according to their stress severity levels. Thus, we obtained human expert rated/annotated examples in order to compare with the xPlNet results. We used a custom built app to enable the expert raters to quickly, and efficiently mark severity measures.

Because ease of use was a priority, we chose to build a standard web application so that users could access the app immediately, on any device, without installation. We also chose not to use any particular frameworks for the front-end or back-end, instead using standard JavaScript and PHP. We used a standard MySQL database to store the marked image data. One developer created an initial prototype of the front-end in a day, which proved very useful for developing a shared vision of the final app (see Fig. 5.17).

The front-end was plain HTML and JavaScript with some AJAX calls into a PHP backend. We used commonly used, well-supported technologies. We used the Raphael JS library to draw vector graphics on top of the leaf images, with simple JavaScript event handlers to simulate drawing on the image. To support touch-based mobile devices, we used the jQuery Touch Punch library to map touch events to mouse events. On the backend, the marked image data were stored by converting the path data to a JSON string and saving the data in a MySQL BLOB-type column. To track multiple users, we had the app open with an enter your user-name field. The user-name was then
included in subsequent back-end requests. Because the app was used only internally and for a very brief time, we did not add any security measures against impersonation.

The web app was deployed on our group-internal web server, which runs a standard LAMP stack (Linux operating system, Apache web server, MySQL database, and PHP). After some brief testing, links to the application and instructions were distributed to experts to begin tagging the images. Tagging was conducted in multiple sessions during the course of one week (see Fig. 5.18).

This procedure was performed on a subset of 1000 random images extracted from the test set data (of 6576 images). The rater-based severity was calculated by marking the diseased part of the leaf and obtaining the ratio of the area (pixel count) of the marked region to the area of the entire leaf. We compare the machine-based explanation maps with human expert ratings (which can suffer significantly from inter-rater variability as shown in Fig. 5.19. Fig. 5.9 presents examples of the explanation framework and expert-rated leaf images for each stressed and healthy cases; we provide more examples in Fig. 5.12.

5.5.2 Failure Cases

Here we mention some of the failure cases that show up while running the framework:

- The presence of shadows and dark spots on the input image: The presence of shadows and dark spots in many examples (even on healthy leaves), posed a challenge for the explanation mechanism in detecting the correct stress signatures (shadow regions are true stress identifiers in some cases), because the trained DCNN model sees the shadow and/or dark regions compared to yellow and brown regions as regions of low interest and thus tends to neglect those regions as untrue stress signatures. However, human raters do not make this mistake.

- The lack of (or unbalanced) focus when capturing the image: Blurry images did not serve as good examples for training or testing the neural network. The architecture in such cases failed to detect proper stress signatures. Stress signatures for blurry (out of focus) images are also difficult for human raters. The problem is exacerbated by the lack of significant amount of symptoms where very small, localized visual cues may get lost.
• The lack of resolution: To address GPU memory issues, the original high-resolution images were resized to low-resolution (64X64) images that were fed into the neural network. This resizing led to information loss and, in some cases, to the model and the explanation mechanism failing to detect the correct stress signatures. The stress signatures of low-resolution images are also difficult for human raters to identify. In our case, high-resolution images were provided to the raters to identify correct stress signatures, whereas low-resolution versions of the same we fed to the neural network to alleviate GPU memory issues. Thus, the lack of resolution posed a problem in several cases when testing the DCNN architecture.

• Incorrect prediction by the CNN model: The presence of all or some of the previously mentioned three causes led to incorrect prediction by the neural network. When incorrect prediction occurred, the explanation mechanism also failed to detect the correct stress signatures on the leaf.

An example for each of the mentioned failure cases is presented in Fig. 5.20.
Figure 5.10: This confusion matrix shows the stress classification results of the DCNN model for eight different stresses and healthy leaves. The overall classification accuracy of the model is 94.13%. The greatest confusion among stresses was found among bacterial blight, bacterial pustule, and Septoria brown spot which can be attributed to the similarities in symptom expression among these stresses.
Figure 5.11: Distributions of spatial correlation between human marked symptoms and machine learning explanations for four stresses: Septoria brown spot, IDC, Herbicide injury and Sudden Death Syndrome. As the distributions are significantly skewed towards high correlation values, they show the success of the DCNN-based severity estimation framework to correctly identify symptoms for these stresses. Few examples with different severity classes (machine learning explanations on left, actual images on right) using a standard discretized severity scale (0-25%: resistant, 25-50%: moderately resistant, 50-75%: susceptible, and 75-100%: highly susceptible).
Figure 5.12: Examples of xPlNet results: Rated leaf images for four different soybean stresses
Figure 5.13: Variation of test accuracy w.r.t. the increasing size of training data
Figure 5.14: Visualizing learnt filters for the input image shown here (Conv. represents a 2D convolutional layer, and Pool. represents a 2D max-pooling layer)
Figure 5.15: xPlNet performance on non-destructive leaf image samples for IDC, Potassium deficiency and SDS; top row shows the original images either with clipboard imaging protocol (IDC) or canopy imaging protocol (Potassium deficiency and SDS); middle row shows the extracted leaf images used for testing; bottom row shows the xPlNet results.
Figure 5.16: This table illustrates the ability of the framework to accurately identify similar stress symptoms used in training the DCNN model, but in non-soybean crops, such as IDC in cucurbits and potassium deficiency in oilseed rape.

<table>
<thead>
<tr>
<th>Non-Soybean Leaves (Images Collected from the Internet)</th>
<th>Disease Names and Labels</th>
<th>Prediction Explanation Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Iron Deficiency Chlorosis (Class 5)</td>
<td>(Highly Susceptible)</td>
</tr>
<tr>
<td></td>
<td>Potassium Deficiency (Class 6)</td>
<td>(Moderately Resistant)</td>
</tr>
</tbody>
</table>

Original Image
Figure 5.17: Design of the web app for expert-based foliar stress identification and severity estimation

Figure 5.18: Illustration showing the tagging process
Figure 5.19: Scatter plot comparing the severity ratings of the same images between two raters for four stresses (IDC, SDS, potassium deficiency and Septoria brown spot) that were pooled and solid red line is the 45° line. The results indicated high inter-rater variability between experienced raters, especially as the stress severity of leaf images increase.
<table>
<thead>
<tr>
<th>Failure Cases</th>
<th>Undesired Condition</th>
<th>Presence of Shadows, gloss and/or Dark Spots</th>
<th>Lack of Focus, Significant Lack of Symptoms</th>
<th>Lack of Resolution</th>
<th>Incorrect Prediction by the CNN model (bacterial spot predicted as bacterial blight, as shown below)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Image</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
<td><img src="image3" alt="Image" /></td>
<td><img src="image4" alt="Image" /></td>
<td><img src="image5" alt="Image" /></td>
</tr>
<tr>
<td>Explanation Output</td>
<td><img src="image6" alt="Image" /></td>
<td><img src="image7" alt="Image" /></td>
<td><img src="image8" alt="Image" /></td>
<td><img src="image9" alt="Image" /></td>
<td><img src="image10" alt="Image" /></td>
</tr>
<tr>
<td>Rater marked Image</td>
<td><img src="image11" alt="Image" /></td>
<td><img src="image12" alt="Image" /></td>
<td><img src="image13" alt="Image" /></td>
<td><img src="image14" alt="Image" /></td>
<td><img src="image15" alt="Image" /></td>
</tr>
</tbody>
</table>

Figure 5.20: Examples in which xPlNet failed to detect proper stress signatures
CHAPTER 6. ENCODING CNN-SURROGATES AS INVARIANCE-CHECKERS IN DEEP GENERATIVE MODELS

In CHAPTER 4, we developed a CNN-surrogate to address the issue of designing active layer morphologies to enhance device performance, especially in OPV applications. The usual methods to quantify morphology are either too costly or too simplistic. Hence we take a data-driven approach - Deep Learning Structure Property (DLSP) development framework to create a morphology quantifier that can perform fast evaluations. We train a custom designed CNN that reads the morphology and classifies it into 10 bins of increasing performance metric $J_{sc}$. Using out-of-sample data sets, we confirm that there is no severe over-fitting issues during the training process.

Here, we try to come up with a generative framework augmented with the developed CNN-surrogate as an invariance checker. We base our framework on Generative Adversarial Networks (GANs) and leverage the ability of the pre-developed DLSP framework to generate microstructures given a target class.

6.1 Introduction

6.1.1 Motivation

Generative Adversarial Networks (GANs) have proven to be highly successful in synthesizing samples arising from complex distributions, including face images (Radford et al. (2016)), content generation (Jin et al. (2017)), image translation (Isola et al. (2017)), style transfer (Zhu et al. (2017)), and many others. However, a well-known issue of GANs is that they incur dramatically high sample complexity; for example, the recent work of Brock et al. (2019) requires 14 million training images and is trained over $\sim 24K$ TPU-hours, which is beyond the reach of normal computing environments.

This challenge partly arises due to the purely unsupervised nature of GAN training. Intuitively, GAN models start learning from scratch, and require lots of training examples before they learn
to reproduce essential features (or *invariances*) present in the training data. However, in several applications, invariances that the generated samples should exhibit are explicitly known prior to training. For example, in scientific simulations, the data samples often obey (universal) laws encoded in the form of well-defined mathematical equations, or obey other statistical or geometric constraints. How, then, should GAN models best leverage such prior knowledge of invariance information?

In (Shah et al. (2019)), the authors propose a systematic extension of GANs that can synthesize data samples obeying pre-specified, analytically defined invariances. This new generative models are termed *Invariance Networks*, or InvNet. This approach subsumes the standard GAN setting; InvNets can learn *implicit* invariances from the training data (similar to GANs), but also enforce *explicit* invariances. Similar to GANs, the InvNet training problem is posed as a minimax game. Training an InvNet requires some care, since interesting new challenges arise. We propose a three-way alternating-optimization style training algorithm that gives useful, stable results across a wide range of invariances and training data. Several experiments are performed and supplemented with theoretical analysis in (Shah et al. (2019)). In particular, the InvNet training problem is studied in simple but illuminating special cases. A (pure) Nash equilibrium analysis for the minimax game in InvNet training is performed, and it is shown that the choice of discriminator architecture plays an influential role in encoding target invariances. InvNet training dynamics are analyzed - this leads the authors to discover that traditional alternating gradient descent updates diverge. As a resolving step, the authors propose training via *extra gradient-descent* and prove its convergence. For a detailed theoretical analysis, please refer to (Shah et al. (2019)) as it is beyond the scope of this chapter.

### 6.1.2 Contributions

In this chapter, we propose a systematic extension of InvNets that can synthesize data samples obeying pre-specified, analytically defined property-based invariances. We call our new generative models *Surrogate Invariance Networks*, or S-InvNet (refer Fig. 6.1). Our approach subsumes the standard InvNet setting; S-InvNets can learn *implicit* invariances from the training data (similar
to InvNets), but also enforce explicit invariances based on a physical property of microstructures using a trained CNN surrogate model, for which this framework has been tailor-made. The primary motivation behind S-InvNet is to generate microstructures based on a target property. Similar to InvNets, the S-InvNet training problem is also posed as a minimax game.

We illustrate S-InvNet performance on a challenging application in computational material science, which is the problem of simulating (binary) microstructures obeying desired properties (Wodo and Ganapathysubramanian (2012a); Torquato (2013)). Here, the equations governing the formation of microstructures are typically very complex, but we require that the target microstructures satisfy an invariance defined by the constraints of the CNN surrogate. This invariance is basically the target class we want our microstructure to belong to. We encode these invariances in the form of matching constraints, and train an InvNet model to generate synthetic microstructures adhering to targets set by the CNN surrogate. We call this new model Surrogate-InvNet or, S-InvNet. We show that S-InvNet significantly outperforms standard numerical microstructure methods in terms of computational costs.

6.2 Backbone to S-InvNet: The InvNet Algorithm

Details of the InvNet algorithm and its theoretical analysis can be found in (Shah et al. (2019)). Nonetheless, we provide a brief overview of the algorithm here, for the sake of completion. Consider a data distribution \( P_{\text{data}} \) defined over set \( D \subseteq \mathbb{R}^d \), and a list of differentiable invariance functions \( \mathbb{R}^d \rightarrow \mathbb{R} : I_i(\cdot), i = 1, 2, ..., r \). The aim of S-InvNet is to generate new samples \( \mathbf{x} \) from \( D \) that satisfy the expression \( I_i(\mathbf{x}) = 0, \forall i = 1, 2, ..., r \). We define our generator to be a function \( G_\theta : \mathbb{R}^k \rightarrow \mathbb{R}^d \) parameterized by \( \theta \). Let \( z \) represent a \( k \)-dimensional latent input vector. In the standard GAN setup (Goodfellow et al. (2014)), the generator is trained by posing a two-player game between a generator \( G \) and a discriminator \( D \), where the discriminator is a function \( D_\psi : \mathbb{R}^d \rightarrow \mathbb{R} \) parameterized by \( \psi \). Using the notation of (Mescheder et al. (2018)), the training objective for GAN is described by:

\[
L(\theta, \psi) = \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} \left[ f(D_\psi(\mathbf{x})) \right] + \mathbb{E}_{z \sim P_z} \left[ f(-D_\psi(G_\theta(z))) \right], \quad (6.1)
\]
for some monotonic function $f : \mathbb{R} \to \mathbb{R}$ and $P_z$ being a known distribution. We focus on Wasserstein GAN (Arjovsky et al. (2017); Gulrajani et al. (2017)), where $f(t) = t$, while noting that our approach below extends to general (differentiable) $f$ functions \textit{mutatis mutandis}. In order to encode the invariances, we propose the following minimax game to train our S-InvNet model:

$$\min_\theta \max_\psi \bar{L}(\theta, \psi), \text{ where } \bar{L}(\theta, \psi) = L(\theta, \psi) + \mu \sum_{i=1}^{n} \mathbb{E}_z [I_i(G_\theta(z))] := L(\theta, \psi) + \mu L_I(\theta). \quad (6.2)$$

We solve the minimax game in a fashion similar to GAN training; we alternately adjust the generator parameters $\theta$ and the discriminator parameters $\psi$ via gradient updates. However, due to the presence of the additional invariance term in $\bar{L}(\theta, \psi)$, we find in practice that a three-way update rule works well: a GAN-like update of $\theta$ via gradient steps of $L(\theta, \psi)$ keeping $\psi$ fixed; a GAN-like update of $\psi$ via gradient steps of $L(\theta, \psi)$ keeping $\theta$ fixed; and an update of $\theta$ via gradient steps of $L_I$. See Alg 3.

**Algorithm 3 Alternating Optimization for InvNet**

**Require:** Set learning rates, termination conditions.

1: \textbf{while} $L_I$ large and $\theta$ has not converged \textbf{do}
2: \hspace{0.5cm} for $l \leftarrow 1$ to $N_G$ \textbf{do}
3: \hspace{1cm} $\theta \leftarrow \theta - \eta_G \nabla_\theta \bar{L}$ \hspace{1cm}$\triangleright$ Generator update
4: \hspace{0.5cm} end for
5: \hspace{0.5cm} for $m \leftarrow 1$ to $N_D$ \textbf{do}
6: \hspace{1cm} $\psi \leftarrow \psi + \eta_D \nabla_\psi \bar{L}$ \hspace{1cm}$\triangleright$ Discriminator update
7: \hspace{0.5cm} end for
8: \hspace{0.5cm} for $n \leftarrow 1$ to $N_I$ \textbf{do}
9: \hspace{1cm} $\theta \leftarrow \theta - \eta_D \nabla_\theta L_I$ \hspace{1cm}$\triangleright$ Generator update
10: \hspace{0.5cm} end for
11: \textbf{end while}

**Role of the input vector $z$.** Unlike the standard GAN setting, we define $z$ as the concatenated vector $[\tilde{z}, c]^T$ with $\tilde{z}$ referring to a random vector sampled from a known distribution while $c$ is a deterministic vector that parameterizes the invariances. This way, $z$ can be used to model both stochastic and deterministic components of the data. Moreover, $c$ gives us additional tuning knobs, allowing a broader range of applications for the S-InvNet model; we elaborate below.

**Variations of alternating optimization.** As mentioned above, we optimize our multi-objective formulation by alternately optimizing over the three sub-components of $\bar{L}$, as presented
in Alg. 3. Our choice of using alternating optimization is informed by insights in recent work (Daskalakis et al. (2017); Mokhtari et al. (2019); Liang and Stokes (2019)) that show that regular gradient descent for minimax games diverges, while methods that take intermediate gradient steps, such as extra-gradient descent, converge to stable Nash equilibria.

**When the invariances perfectly define the target distribution.** In cases where the set of invariances, \( I_i \), are necessary and sufficient conditions for specifying \( D \), the invariances themselves can be interpreted as serving the role of the optimal discriminator. This has two major implications: (i) \( D_{\psi} \) is constant, Lines 5-7 of the algorithm are no longer needed, and 6.2 is a regular minimization problem that can be stably solved; (ii) more importantly, since the gradients of \( \bar{L} \) no longer depend on the data distribution \( P_{data} \), InvNet training can be performed in a data-free manner. This property is crucial in modeling complex physical systems that are fully described through their governing PDEs.

### 6.3 S-InvNet: Generating Microstructures using CNN-surrogate as Invariance to InvNet

In computational material science, material distribution is often represented by an image describing the arrangement of constituents within a material. The statistical properties of such images govern the physical properties of the underlying material. Synthesizing microstructures adhering to specific statistical properties is, therefore, a crucial component of new material discovery.

We focus on binary microstructures (corresponding to black/white images) exhibiting phase separation; their dynamics are governed via the well-known Cahn-Hilliard (CH) equation (Cahn and Hilliard (1958)). This is a fourth-order nonlinear PDE, and its solution requires a significant amount of compute time.
Figure 6.1: S-InvNet: The proposed architecture for online targeted microstructure generation. The CNN surrogate (working in tandem with the Discriminator) is used as the invariance checker which ensures that the Generator output belongs to the target class.

We remedy this by training an S-InvNet to generate microstructures adhering to desired class values. We encode the desired microstructureal properties (which is directly related to the target class) into the InvNet formulation using the CNN-surrogate as the invariance:

$$L_I = \lambda_1 \| \text{Pr}(G_\theta(z)) - \text{Pr}(Y_{target}) \|^2_2$$  

(6.3)

where $\text{Pr}(G_\theta(z))$ represent the CNN-surrogate prediction for the fake data generated by the Generator, and $\text{Pr}(Y_{target})$ is the prediction for the target class (for high performing morphologies, this target class is class 9).
Figure 6.2: S-InvNet II: We modify the previous S-InvNet architecture by utilizing the trained surrogate as a filter to the true microstructure data, before it is passed to the Discriminator. The surrogate allows only the target data (i.e., the data belonging to the target class) to be passed to \( D(\cdot) \)

For training the S-InvNet, we use a publicly available data set of 2D binary microstructures containing \( \sim 34k \) images (Shah et al. (2019); Pokuri et al. (2018a)). We then follow a similar process where we alternately optimize over the generator loss, the discriminator loss and the invariance loss as in alg:aio.

### 6.4 Results

The results in Fig. 6.4 to Fig. 6.8 show the generated images adhering to target invariances (classes) along with the corresponding classification predictions of the set of S-InvNet generated images for target classes 7 to 9 respectively. We also note that the training time incurred by S-InvNet
Figure 6.4: Samples from Surrogate-InvNet (S-InvNet) generated microstructures for target class 7 and corresponding predictions (population size: 64 samples)

Figure 6.6: Samples from Surrogate-InvNet (S-InvNet) generated microstructures for target class 8 and corresponding predictions (population size: 64 samples)

is amortized over the generation time required to simulate many such microstructures; to generate a fixed number of microstructures InvNet already obtains speedups over existing numerical solutions. A key point to note here is that for training the GAN part of S-InvNet with images belonging to the target class quite heavily ensures that the output images belong to the target class alone. In such a case, having the CNN surrogate as the invariance checker is not a necessary condition for training and obtaining a successful model, as the Discriminator quite easily ensures that the Generator outputs strictly belong to the target class. However, the complexity of the problem lies in the fact whether or not we could do the same even when the Discriminator is fed with images of random classes or of totally opposing target classes - consider for instance, the following scenario - the Discriminator within S-InvNet being fed with images belonging to class 0 - the lowest performing
(a) Samples of generated microstructures for the target class. (b) Histogram showing surrogate predictions of generated microstructures. Here, the horizontal axis denotes the class value.

Figure 6.8: Samples from Surrogate-InvNet (S-InvNet) generated microstructures for target class 9 with corresponding predictions (population size: 64 samples)

![Image](image1.png)

(a) Histogram showing surrogate predictions of generated microstructures. Here, the horizontal axis denotes the class value. Training data consisting of Classes 1, 2, 8 and 9.
(b) Histogram showing surrogate predictions of generated microstructures. Here, the horizontal axis denotes the class value. Training data consisting of Class 0 only.

Figure 6.10: Samples from Surrogate-InvNet (S-InvNet) generated microstructures for target class 9 with variations in presence of training data classes (population size: 64 samples)

class, while the invariance checker i.e., the CNN surrogate is trying to ensure that the output belongs to class 9. What then? We tried out this scenario as well, as shown in Fig. 6.10.

**Observations from Fig. 6.10** As we see here, in both cases, although the target class was 9 i.e., the CNN surrogate was to enforce the class 9 as the invariance, we tend to have mixed outputs for the model trained with mixed classes 1, 2, 8 and 9 (Fig. 6.10 (a)) with more than 50% of it being predicted as a higher class morphology. Also, as Fig. 6.10 (b) shows, for the model trained with class 0 morphologies only, the overall performance of the model is poor i.e., the CNN surrogate fails to enforce the invariance and most of the outputs are shown to belong to class 1 and a few of them class 2 and class 3 with noen of them being high performing morphologies even though the target class has always been class 9.
Inference from Fig. 6.10 The observations we have made (as discussed in the previous paragraph) leads us to infer that the Discriminator is playing a major role instead of the CNN surrogate as the invariance checker during the training phase. There could be a few ways around this problem. We propose an immediate solution to this problem in Section 6.5. Further investigations have been left as a future extension of the current work presented in this chapter. A few ideas that have potential have been put forth in CHAPTER 8.

6.5 S-InvNet II

Following our observations from Fig. 6.10, we tweaked our initial S-InvNet model (refer Fig. 6.1) and come up with yet another framework, which we simply name S-InvNet II (refer Fig. 6.2). The primary difference between this and our initial framework is that the training data before being passed to the Discriminator, is fed into our Invariance checker (the trained surrogate model), which then acts as a filter. The purpose of this filter is to pass only the target class data to the Discriminator and block passage of any other data that doesn’t belong to the target class. This way, even if the training data has multiple classes within its pool, the framework is able to handle the presence of class disparities well.

Results from S-InvNet II As we see from Fig. 6.12, S-InvNet II works quite well in generating the target class data (class 9). We pass training data consisting of classes 1, 2, 8 and 9 and the
surrogate model seems to successfully vet out the non-target data (classes 1, 2 and 8) as is evident from the results (compare Fig. 6.10 (a) vs Fig. 6.12 (b)). S-InvNet II and its variants thereof would thus be the choice architecture for this problem.
Surrogate modeling is computationally attractive for problems that require repetitive yet expensive simulations, such as deterministic design, uncertainty propagation, optimization under uncertainty or inverse modeling. Data-efficiency, uncertainty quantification and generalization are the main challenges that face surrogate modeling, especially for problems with high dimensional stochastic input, such as material properties (Bilionis et al. (2013)), background potentials (Charalampidis et al. (2018)), etc.

This work focuses on physics-constrained surrogate modeling for PDEs with high-dimensional spatially-varying coefficients without simulation data. We first show that when solving deterministic PDEs, the CNN based parameterizations are more computationally efficient in capturing multiscale features of the solution fields. Furthermore, we demonstrate that the proposed method achieves high predictive performance, despite the fact that it does not make use of any output simulation data. In addition, it produces just as good predictions under extrapolative conditions as when out-of-distribution test input data sets are used.

Rest of the chapter is organized as follows. Section 2 provides the definition of the problems of interest including the solution of PDEs, surrogate modeling and uncertainty quantification. Section 3 provides the parameterization of the solutions with FC-NNs and CNNs, the physics-constrained learning of a deterministic surrogate and the variational learning of a probabilistic surrogate. Section 4 investigates the performance of the developed techniques with a variety of tests for various PDE systems. We conclude in Chapter 4 with a summary of this work and extensions to address limitations that have been identified.
7.1 Network Architecture

The network in Fig. 7.1 and Fig. 7.2 is made up of a series of convolution and maxpooling layers whose feature maps perform the detection of image and feature edges, shapes and objects. Convolutions are initiated by choosing appropriate kernel sizes and numbers that are shifted and convolved on the input images, following which, the ReLU activation (Nair and Hinton (2010)) is applied for learning the joint kernels in local neighborhoods that are useful for describing the explanatory features at that layer. Maxpooling layers are included to learn the scale invariance of the input images. The fully connected layers embed the high dimensions in a low dimension space, activates ensure that the orientations do not necessarily hurt the network performance and produces the classification of the network at the coding layer with a softmax activation (Hinton and Salakhutdinov (2009)). The architecture uses the Encoding and Decoding layer (as shown in Fig. 2.11) as a unit to generate the U like structure that approximates the PDE using the Physics-constrained losses.

![Figure 7.1: Network architecture: DenseED. Figure design idea sourced from Zhu and Zabaras (2018)](image)

**Related work.** Neural networks are increasingly being used as functional surrogates in modeling complex physical systems. Point-wise neural network methods - Raissi et al. (2017b,d); Pang et al.
Figure 7.2: Dense convolutional encoder-decoder network as the deterministic surrogate. The models input is the realization of a random field, the models output is the prediction for each input field. The model is trained with physics-constrained loss without target data. Figure design idea sourced from Zhu et al. (2019).

Zhu and Zabaras (2018); Zhu et al. (2019) employ the use of a convolutional Encoder-Decoder (ED) as a PDE surrogate using physics-informed invariances. We extend this Encoder-Decoder network to similarly solve PDEs with spatiotemporal variances and at the same time, propose and employ a novel alternating loss minimization method for the ED network as well.

### 7.2 Problem Definition

#### 7.2.1 Problem Statement

We focus on solving a PDE which varies both with space and time, namely the Burgers Equation and employ a Encoder-Decoder (ED) Network to act as a surrogate for the PDE by optimizing the Physics-Driven loss terms as discussed in the subsequent sections. While ED networks have previously been used for solving spatially varying systems, this to our knowledge would be the first time such a technique has been used to solve a complex PDE like Burgers which varies with both space and time. We also extend our approach to propose a novel method of optimizing 2 different loss terms alternately to try to achieve similar or better results.
We consider the deterministic surrogate introduced in Zhu et al. (2019) for our case i.e., solving Burgers’ Equation, where input to the framework are samples of input fields that obey certain initial and boundary conditions as shown in Fig. 7.3. This field is taken as an input by the ED which is then optimized (solved) using a joint ’Physics-Constrained’ loss term, governed by the governing Burgers PDE given by \( \text{Loss}_{PDE} + \lambda \times \text{Loss}_{Boundary} \), where \( \lambda \) is the weight (Lagrange multiplier) that softly enforces the boundary conditions.

Figure 7.3: Inputs to the Model - 2 different cases

### 7.2.2 Generating Solutions for PDEs

We demonstrate that the solution set of partial differential equations (PDEs) describing dynamics of physical systems can be accurately recovered using an ED model. Given the parametric form of a system of PDEs, its solution set is governed by the given physical constants (parameters) together with any boundary conditions. ED provides an alternate, data-free approach to solving PDEs. The key idea is that since the PDE itself serves as the optimal discriminator and hence, no training data is required.

We demonstrate this via a simple non-linear PDE called Burgers’ Equation (Bateman (1915)). We use a conditional input to ED to control the solution set of Burgers’. Therefore, unlike Zhu et al. (2019), a single well-trained ED can generate solutions corresponding to a variety of boundary...
conditions as well as varying physical parameters. Before we describe our approach, we provide
a quick PDE primer. The solutions to 1d, time varying PDEs can be represented as images by
choosing \( x \) and \( t \) as two axis of the image.

**Inviscid Burgers’ equation.** This is a non-linear PDE encountered in fluid mechanics and
nonlinear acoustics. The inviscid form of the Burgers’ equation, defined in equation 7.2, assumes a
non-diffusive fluid through which a wave with initial state \( f_i \) is passed. Let \( U(x, t) = [u_0, u_1, \ldots, u_n] \)
be a particular solution of the Burger’s equation. Then,

\[
\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0 \tag{7.1}
\]

which can be re-written as:

\[
U_{x,t} + U \odot U_{x,x} = 0; \quad U(x, 0) = f_i(x), \tag{7.2}
\]

where the partial derivative with respect to \( x \) is represented as \( U_{x,x} \).

Suppose we model the field \( U(x, t) \) as an image (where rows correspond to space and columns
correspond to time; see Fig. 7.3 and Section 7.4).

The boundary condition is a structural invariance that we enforce by minimizing the \( \ell_2 \) loss
between the boundary of the generated solution \( U(x, 0) = \mathbb{P}_\Omega(N_\theta(z)) \) and the given boundary . In
our experiments, we set \( b \) as a (discretized) raised-cosine function parameterized on \( c \). We sample
\( b \) uniformly from set \( B \) given as:

\[
B = \{x|x = \frac{1}{2}(1 - \cos(2\pi z/d)), \quad z \in [2.0, 4.0, 5.5, 6.0]\}\}. \tag{7.3}
\]

Our ED minimizes (a weighted combination of as well as alternately) the following losses:

\[
L_R = \mathbb{E}_z \left[ \left| \left| N_\theta(z)_{x,t} + N_\theta(z) \odot N_\theta(z)_{x,x} \right| \right|_2^2 \right], \quad L_I = \mathbb{E}_z \left[ \left| \left| \mathbb{P}_\Omega(N_\theta(z)) - b \right| \right|_2^2 \right] \tag{7.4}
\]
We calculate the partial derivatives by convolving the generated image with the directional Sobel operators (Sobel and Feldman (1968)) with respect to the $x$ and $t$ axes, reducing the border effects by replication. **Viscid Burgers’ equation.** We extend the above algorithm to generate solutions for a family of PDEs, with a physical scalar parameter indexing each PDE. We consider the family of viscid Burger’s equation represented in equation 7.6 with the viscosity term, $\nu$:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$  \hspace{1cm} (7.5)

which can be re-written as:

$$U_x + U \odot U_x = \nu U_{x,x}.$$  \hspace{1cm} (7.6)

Similar to the inviscid case, we provide the boundary condition $b$ as input to the network and the coefficient of viscosity $\nu$ as input to the loss function. Our input vector for testing conditions becomes $z = c = [b, \nu]$, where $b$ is sampled uniformly from $B$ (equation 7.3), and $\nu$ is sampled from the set $[0, 0.01, 0.0225, 0.0725, 0.1]$. Section 7.4 depicts the solutions corresponding to different combinations of $b$, $\nu$ and $\lambda$ (the Lagrange weight multiplier). We test with $\lambda$ values of 1 and 10 for the Summed Loss and $\lambda = 1$ for the Alternating version.

We compare our ED-based surrogate solutions to numerical solutions (computed using explicit Euler methods) for both the viscid and inviscid cases and observe that our model is fairly accurate. Moreover, ED shows effective generalization by generating solutions for boundary conditions that were not used during training as well. An important thing to note here is that ED generalizes as well. The model is able to generate a solution for the initial condition, $u_0 = \frac{1}{2}(1 - \cos(\frac{2\pi x}{a}))$ in-spite of not being trained on it.

### 7.2.3 Data-set Generation

The training data for the surrogate model of interest here include the realizations of the random input field and the corresponding multi-output obtained from simulation. Of interest is to address problems with limited training data sets considering the high-computational cost of each simulation.
run. Note that in this work our focus is on the image-to-image mapping and its performance on uncertainty quantification tasks. We will thus assume that enough input samples are provided both for testing and output statistics calculation.

We generate 64 random realizations based on the governing PDE equation for 2 different initial conditions. This randomization is done for every training iteration. While we randomize the training data set to include all possible cases of initial conditions (with 64 randomizations), we keep the validation data fixed to \( z = 2, 4, 5.5, 6 \) and take note of how these 4 conditions evolve as training progresses.

### 7.3 Evaluation Metric: Relative \( \ell_2 \) Error

We use the relative \( \ell_2 \) error as the metric of choice to evaluate the performance of the ED-network. It is defined as follows:

\[
\epsilon = \frac{1}{T} \sum_{i=1}^{T} \left( \frac{||\hat{y}_i - y_i||_2}{||y_i||_2} \right)
\]  

(7.7)

where \( T \) is the total number of test inputs, \( \hat{y}_i \) is the surrogate prediction and \( y_i \) is the exact numerical solution. \( ||.||_2 \) is the \( \ell_2 \) norm. Adam (Kingma and Ba (2014b)) is used as the optimizer with learning rate of 0.0003 for both the alternating as well as summed loss versions of the framework.

### 7.4 Results for ED framework optimized with with Combined Loss, \( \mathcal{L} \)

Section 7.5 presents a comparison between the alternating loss and the combined loss minimization versions for our ED network. We note that combined loss optimization performs better (even if marginally) in most of the cases. As such, we present several experimental conditions using the combined loss optimization method in this section. The Combined Loss optimization for the ED network is performed using equation 7.8.

\[
\mathcal{L} = L_R + \lambda L_I
\]  

(7.8)
**Organization** This section is organized as follows: Section 7.4.1 shows results for viscid flow for $\lambda = 1$ where Fig. 7.4 to Fig. 7.7 and Fig. 7.10 to Fig. 7.21 show results obtained from the ED network and the its comparison with the exact solution, Fig. 7.8 and Fig. 7.9 show the temporal variation for the predicted solution for Input Field 1 and Input Field 2 respectively for $\lambda = 1$. Section 7.4.2 shows results for viscid flow for $\lambda = 10$ where Fig. 7.22 to Fig. 7.25 and Fig. 7.28 to Fig. 7.39 show results obtained from the ED network and the its comparison with the exact solution, Fig. 7.26 and Fig. 7.27 show the temporal variation for the predicted solution for Input Field 1 and Input Field 2 respectively for $\lambda = 10$. Section 7.4.3 and Section 7.4.4 show results for inviscid flow for $\lambda = 1$ and $\lambda = 10$ respectively.

### 7.4.1 Results for Viscid Flow, $\lambda = 1$, Combined Loss function

This section presents the results obtained from the ED network and the its comparison with the exact solution for Viscid Flow for $\lambda = 1$. 
Figure 7.4: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row) - leftmost column shows the Input Field (the input to the ED framework), middle column shows the ED network solution and the rightmost column shows the relative $\ell_2$ error between the Exact numerical solution and the ED output. Parameters: $[\lambda = 1, \nu = 0.01, z = 2]$
Figure 7.5: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.01, z = 4]$
Figure 7.6: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.01, z = 5.5]$.

Figure 7.7: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.01, z = 6]$.
Figure 7.8: Temporal variation for the predicted solution for Input Field 1 with parameters $\lambda = 1, \nu = 0.1$ - time axis corresponds to a range of $t = 0 - 0.2s$

Figure 7.9: Temporal variation for the predicted solution for Input Field 2 with parameters $\lambda = 1, \nu = 0.1$ - time axis corresponds to a range of $t = 0 - 0.2s$
Figure 7.10: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0225, z = 2]$

Figure 7.11: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0225, z = 4]$
Figure 7.12: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0225, z = 2]$.

Figure 7.13: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0225, z = 6]$.
Figure 7.14: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0725, z = 2]$.

Figure 7.15: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0725, z = 4]$.
Figure 7.16: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0725, z = 5.5]$.

Figure 7.17: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.0725, z = 6]$.
Figure 7.18: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.1, z = 2]$

Figure 7.19: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.1, z = 4]$
Figure 7.20: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.1, z = 5.5]$
Figure 7.21: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0.1, z = 6]$
7.4.2 Results for Viscid Flow, $\lambda = 10$, Combined Loss Function

This section presents the results obtained from the ED network and the its comparison with the exact solution for Viscid Flow for $\lambda = 10$.

Figure 7.22: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.01, z = 2]$

\[ \text{Relative } l_2 \text{ error} = 0.3594 \]

\[ \text{Relative } l_2 \text{ error} = 0.0392 \]
Figure 7.23: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.01, z = 4]$.

Figure 7.24: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.01, z = 5.5]$.
Figure 7.25: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.01, z = 6]$

Figure 7.26: Temporal variation for the predicted solution for Input Field 1 with parameters $\lambda = 10, \nu = 0.1$ - time axis corresponds to a range of $t = 0 - 0.2s$
Figure 7.27: Temporal variation for the predicted solution for Input Field 2 with parameters $\lambda = 10, \nu = 0.1$ - time axis corresponds to a range of $t = 0 - 0.2s$

Figure 7.28: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0225, z = 2]$
Figure 7.29: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $\lambda = 1, \nu = 0.0225, z = 4$

Figure 7.30: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $\lambda = 10, \nu = 0.0225, z = 5.5$
Figure 7.31: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0225, z = 6]$.

Figure 7.32: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0725, z = 2]$.
Figure 7.33: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0725, z = 4]$

Figure 7.34: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0725, z = 5.5]$
Figure 7.35: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.0725, z = 6]$

Figure 7.36: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.1, z = 2]$
Figure 7.37: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $\lambda = 10, \nu = 0.1, z = 4$

Figure 7.38: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $\lambda = 10, \nu = 0.1, z = 5.5$
Figure 7.39: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0.1, z = 6]$
7.4.3 Results for Inviscid Flow, $\lambda = 1$, Combined Loss Function

This section presents the results obtained from the ED network and its comparison with the exact solution for Inviscid Flow for $\lambda = 1$.

Figure 7.40: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0, z = 2]$
Figure 7.41: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0, z = 4]$

Figure 7.42: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0, z = 5.5]$
Figure 7.43: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 1, \nu = 0, z = 6]$
7.4.4 Results for Inviscid Flow, $\lambda = 10$, Combined Loss Function

This section presents the results obtained from the ED network and the its comparison with the exact solution for Inviscid Flow for $\lambda = 10$

Figure 7.44: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0, z = 2]$
Figure 7.45: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0, z = 4]$

Figure 7.46: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0, z = 5.5]$
Figure 7.47: Predicted solutions generated using the ED (Encoder-Decoder) network for Input Field 1 (top row) and Input Field 2 (bottom row). Parameters: $[\lambda = 10, \nu = 0, z = 6]$
We note that for most cases, the value of Lagrangian, $\lambda = 1$ performs better than $\lambda = 10$ (Relative $\ell_2$ error is higher in case of $\lambda = 10$ than $\lambda = 1$). Thus, imposing a higher weight on the initial condition Loss term, $L_I$, does not help the network model the physics well.

### 7.5 Comparison between Results for ED framework optimized with with Alternating and Summed Loss terms

Fig. 7.48 and Fig. 7.49 show comparisons between Alternating and Summed Loss versions for the solutions generated by the ED network for Input Field 1 and Input Field 2 respectively. We see that both alternating (say, case A) and summed (say, case B) loss outputs are comparable to each other qualitatively as well as quantitatively (loss values). However, for most cases, we also note that the summed loss outputs perform marginally better. This can be attributed to the more grainy output obtained from the network in case of alternating loss. Thus, even though the network predicts the flow well in both cases - since, the output is grainy in the first case, the error values are higher. This grainy output can be attributed to sub-optimal parameters for the alternating version. While the same parameters were used for both cases, this leads us to propose that further hyperparameter search is necessary for the alternating loss case in order to train a more robust model. We compare the versions for $\lambda = 1$ only. An interesting point to note for the alternating optimization step is that it reaches the optimal solution point for the PDE at a much faster pace compared to the summed loss optimization. Where the summed loss achieves the mentioned accuracy levels (refer to Fig. 7.48 and Fig. 7.49) at the 15,000$^{th}$ iteration, similar accuracy levels are achieved much earlier at the 2,000$^{th}$ iteration by the alternating version.
Figure 7.48: A comparison between absolute errors between ED output and exact numerical solution for combined loss and alternating loss optimization for Input Field 1. The values beneath each sub-figure represent the relative $\ell_2$ error between the network output and the corresponding exact solution.

Input Field 1, $\lambda = 1$
Figure 7.49: A comparison between absolute errors between ED output and exact numerical solution for combined loss and alternating loss optimization for Input Field 2. The values beneath each sub-figure represent the relative $\ell_2$ error between the network output and the corresponding exact solution.
CHAPTER 8. CONCLUSION AND FUTURE DIRECTIONS

So far in the previous chapters, we have discussed real-world problems where we have added substantial merit and novelty to existing Deep Learning frameworks and in the process, augmenting existing algorithms and coming up with new ones to make them much more stable and robust to frequent changes, make them trustworthy and reliable when handling specific tasks. We have also introduced weak-supervision for specific tasks such as counting of crop flowers and predicting crop yields using such counts. We also come up with ways to incorporate actual physical properties with Deep Learning frameworks and provide alternative, fast accurate data-driven ways to existing methods that could potentially replace existing age-old techniques which have been used till date. Furthermore, we propose Neural Network (CNN) driven approaches in solving PDEs using novel physics-driven loss functions and optimization methods and extend existing literature that deal with simple spatially varying PDEs to more complex equations that evolve both with space and time.

This thesis thus, follows a pattern of dealing with several aspects that come up within existing engineering systems and proposes to provide alternative, fast and effective solutions to existing problems when it comes to accuracy, speed, trustworthiness and robustness of implementing Deep Learning based methods to these systems. We now provide a few extensions to the work that has been presented in each of the preceding chapters to bring this chapter to a conclusion:

8.1 Future Directions

We propose a few future directions extending from the work presented in CHAPTER 3. These may be (note that this list is not exhaustive):

- Evolving the current weak supervision strategy to a fully functional active-learning methodology,
- Possibilities of real time counting by deploying the trained model on UAVs/ground robots and smart-phones,

- Incorporating further diversity (e.g., white-sorghum head images) into the training data-set,

- Developing a domain knowledge-based strategy for automated and smart selection of “distinct” training images. This stems directly from the final point made in the conclusion section.

From CHAPTER 4, some concepts that demand interest are questions of how to integrate a physical phenomena of choice into the training process of the Deep learning Structure Property (DLSP) framework. Can these physics based intuitions (as discussed in the chapter) can be exploited to reduce the demand on the size of data for training? Can a most effective data set be created to reduce the size of the training data? Can we make the training process more robust to adversarial attacks? All these questions form the scope of future study.

In CHAPTER 5, we discuss some of the failure cases that the proposed framework presents in 5.5.2. A suitable future direction would be to investigate these cases further and come up with potential solutions.

CHAPTER 6 presents us with a framework that enables property-based microstructure design by combining a generative and a classification model, where we use the classification model as an invariance checker. While we note that the hybrid model, S-InvNet is successful to a certain extent, it presents problems in form of the Discriminator overpowering the CNN surrogate as its invariance enforcer. A future direction to explore based on these findings would be to look at how to make the Discriminator and the CNN invariance checker to work in tandem to produce meaningful and desirable microstructures as per the given target class value. One potential scenario would be something like this - The training data set has both classes 9 and 2 - the way S-InvNet is right now, it generates microstructures both from class 9 and class 2, even though the invariance checker (the CNN surrogate) only asks for class 9. The key idea is to also show the training data to the invariance checker before passing it to the discriminator part of S-InvNet (which we do by proposing the S-InvNet II architecture, however, this is just an initial step and the framework has a lot of
scope for improvement and fine-tuning). An extreme case would be to filter out the real (training) data that does not pass the invariance check (class 2 data in this case). Thus, these (the class 2 data) do not get exposed to the discriminator and only those that pass the invariance check are. A more elegant approach would be to weigh the real (training) data in a certain way such that sample priority is inversely proportional to the invariance loss, i.e., high loss samples (class 2 in this case) should be given less importance by the discriminator and vice-versa.

In CHAPTER 7, we set out to propose that a suitable neural network driven approach can be used in an unsupervised manner (without using labelled data) to solve PDEs and in the process, provide feasible and meaningful solutions to complex physical systems governed by such PDEs, in the process, taking into account the underlying physics governing such systems. We extend existing literature which has done work on very simple PDEs (only varying in the spatial domain - Darcy FLow) and extend it to the spatial as well as temporal domain (Burgers’ equation). Furthermore, we propose a novel approach - optimizing the neural network by using an alternating loss minimization approach. We note that the ED network is successfully able to model the solution of the PDE (Burger’s equation) without strictly using labelled data. On the contrary it leverages the intrinsic information (Physics) embedded within the Physical governing equation that governs the time-varying flow to effectively solve the problem. This leads us to conclude that the ED network can be a suitable candidate for a surrogate model when solving such equations. As an extension to the work already presented in this chapter, we first propose to further optimize the ED network for the alternating minimization step to solve the PDE in question without exhibiting any of the grainy artifacts that present themselves in the current solutions. This involves a hyperparameter search for the network and a suitable λ value. Thus, a more rigorous analysis into the dynamics of such cases of training the ED network (especially for the alternating loss case) is warranted. We also note that the relative error rates are higher especially for the inviscid case (for Input Field 1) as well as when z value is increased (again, this is most prominent for Input Field 1). This can be attributed to the grainy artifacts within the output itself which could be a possible output of
feature approximation error. Again, exploring the networks hyperparameters and the choice loss function can be a potential solution and future work direction.

Deep Reinforcement Learning (RL) has also been used to develop methods to imitate complex physical motion (Peng et al. (2018)). We see that motion-based systems can be modeled efficiently using Deep Learning techniques. Even before the boom of recent Deep Learning based methods, motion strategies could be modeled using hard-coded physics-based controls. Deep Learning based methods only makes this modeling easier. Deep Reinforcement Learning uses a reward-based learning strategy that aims to reach a certain goal or end state where reward is maximum and penalty is minimized. Although purely Physics-based models are yet to be incorporated along with Deep RL based methods, this could be an exciting new research direction to keep in mind while moving forward.

An interesting point to note from the works presented in CHAPTERs 6 and 7 is that coupling physics-based models with Machine Learning and Deep Learning based methods help in simplify design of complex experiments as well and in some cases do away with it altogether. This has a potential to be a big leap forward towards fast generation of sample data for many real-world simulations and experiments which otherwise have been difficult to carry out because of physical limitations. Deep Learning thus can provide solutions to previously non-solvable or non-verifiable problems. A few instances where such methods can be used are structure-property explorations, fault detection, automated control development and structural health monitoring applications. Exploring Uncertainty Quantification for spatially and temporally varying non-linear stochastic PDEs is also of interest and can be a well sought-after problem to solve using ED networks and/or Invariant Networks.

Overall, we propose several new avenues based on Deep Learning that encompasses and aims to solve different aspects and existing problems of human engineered systems. We further propose directions to extend what has already been presented in earlier chapters that constitute this thesis which brings the present chapter to a conclusion.
REFERENCES


