DeepSoil:

A Deep-Learning Framework for Rapid Low-Cost Estimation of Soil Particle Size Distributions from Digital Microscope Images

A thesis submitted to McGill University in partial fulfillment
of the requirements of the degree of

Master of Science

Jeffrey Spiers

Department of Bioresource Engineering
McGill University, Montreal, Quebec, Canada

December 2020

Abstract

Because soil particle size distribution, also known as soil texture, exerts a strong influence over soil properties of interest to scientists and agricultural producers alike, accurate methods for its measurement are desirable. Existing methods for assessing soil texture rely on either labour-intensive laboratory analysis or costly instruments. Images of soil samples captured by comparatively inexpensive digital microscopes may be analyzed using deterministic or adaptive computer vision algorithms. Deep learning models such as convolutional neural networks (CNNs) can be used to extract features from images and learn intricate data representations from which to make accurate predictions.

This research investigated the application of CNNs to the prediction of soil texture. A deep learning system was developed to train and test multiple CNN models in order to infer soil texture from images of soil samples captured using a digital microscope. Training samples were labeled with reference soil texture values measured using the hydrometer method. The best-performing model predicted sand, silt, and clay content with mean absolute errors of 9.2%, 6.2%, and 5.3%, respectively, and mean absolute error of the ℓ^2 norm of 13.2%. Median absolute prediction errors were 7.1%, 4.8%, and 3.8% for sand, silt, and clay, and 10.8% for the ℓ^2 norm. The coefficient of determination (R^2) statistic for the soil texture predictions was 0.66 for the ℓ^2 norm, with R^2 values for the individual sand, silt, and clay components of 0.68, 0.68, and 0.55, respectively.

Keywords: deep learning; convolutional neural networks; computer vision; image texture analysis; machine learning; precision agriculture; soil particle size analysis; soil texture; digital microscopy; cloud computing.

Résumé

La distribution granulométrique du sol, aussi appelée texture du sol, a une influence importante sur des propriétés du sol qui intéressent autant les scientifiques que les producteurs agricoles. Des méthodes précises de granulométrie du sol sont donc souhaitables. Les méthodes actuelles dépendent soit d'une analyse de laboratoire exigeante ou d'instruments coûteux. Des images d'échantillons du sol capturées par un microscope numérique relativement abordable peuvent être analysées par des algorithmes de vision par ordinateur soit déterministes ou adaptifs. Les modèles d'apprentissage profond et plus particulièrement les réseaux de neurones convolutifs (RNC) peuvent extraire des éléments des images, apprendre des représentations complexes et formuler des prédictions précises.

Ce projet de recherche a étudié l'application des RNC afin de prédire la texture des sols. Un système d'apprentissage profond a été développé pour entraîner et tester plusieurs modèles RNC afin de prédire la texture des sols à partir d'images d'échantillons de sols capturés avec un microscope numérique. Les données de formation ont été annotées avec des valeurs références mesurées par hydromètre. Le modèle le plus performant a prédit le contenu de sable, limon, et argile avec erreur absolue moyenne de 9,2%, 6,2%, et 5,3%, respectivement, et une erreur absolue moyenne ℓ^2 de 13,2%. L'erreur absolue médiane était de 7,1%, 4,8%, et 3,8% pour le sable, le limon, et l'argile, et de 10,8% pour la norme ℓ^2. Le coefficient de détermination (R^2) des prédictions de texture du sol étaient 0,66 pour la norme ℓ^2 et 0,68; 0,68; 0,55 pour le sable, le limon, et l'argile.

Mots-clés: apprentissage profond; réseaux de neurones convolutifs; vision par ordinateur; analyse de texture; apprentissage automatique; agriculture de précision; analyse de distribution granulométrique; texture du sol; microscopie numérique; informatique en nuage.

Dedication

The United Nations' Food and Agriculture Organization (FAO) has estimated that over 25% of the world's population marginally survives by producing their own food on small land plots, with many suffering poverty and food insecurity [1]. This dissertation is dedicated to these subsistence farmers, whose daily toil is beyond anything I know.

Precision agriculture technologies often cater to the needs of wealthier producers. May those of us fortunate enough to have acquired knowledge and skills in this field also devote ourselves to helping those who have the most left to gain.

Acknowledgements

Thank you first of all to A&L Canada Laboratories Inc. for partnering with the McGill Precision Agriculture and Sensor Systems (PASS) research team and providing most of the valuable data on which this research was based. Special thanks to Dave Stallard for coordinating and performing the time-consuming work of capturing images of hundreds of soil samples and regularly uploading them and their associated metadata to the PASS Lab FTP server.

Thank you also to the National Science and Engineering Research Council (NSERC) for their financial support.

Thank you to Dr. James J. Clark and Dr. Tal Arbel of McGill's Centre for Intelligent Machines and Department of Electrical and Computer Engineering for their truly excellent teaching. Dr. Clark's *Computational Photography* course in the fall semester of 2018 provided me with a strong foundation in camera optics, digital imaging, and computer vision algorithms, and Dr. Arbel's advanced course in *Statistical Computer Vision* in the winter of 2019 further strengthened my theoretical knowledge of Bayesian statistics, regularization, information theory, Markov random fields (MRFs), and a wide range of related topics. Perhaps most importantly, the rigorous assignments and term projects for these graduate courses challenged me to sharpen my Python programming skills and introduced me to the OpenCV computer vision and NumPy numerical computing libraries, laying the foundation for the thesis work in applied deep learning presented herein.

Thank you also to my fellow students in the PASS research team for their great help, support and inspiration. Thank you Maxime Leclerc, Roberto Mario Buelvas, and Mohammed Debbagh for introducing me to the PASS Lab's computer-aided design (CAD) and 3D-printing tools and for assisting in the design and machining of various hardware accessories for the microscope holder. Thanks also to Marie-Christine Marmette and

Amanda Boatswain Jacques for their helpful counsel and cheerful guidance as fellow Master's students.

Thank you to my parents, siblings, nephews and niece for supporting me not only during this degree, but throughout all the preceding years of my life. I am incredibly privileged to have their constant wind in my sails.

Thank you to my dear friends in Montreal, across Canada, and elsewhere for checking in on me and occasionally drawing me out of the reclusive world of engineering research to engage in real live human interactions with real live humans. Special thanks to friend and former colleague Dr. Julien Lamoureux for his infectious positive attitude, exemplary self-discipline, and creative advice, and for helping to resurrect my engineering career.

Thank you to Dr. Asim Biswas for his pioneering contributions in applying computer vision techniques to soil particle size analysis and for his guidance in matters of soil science while serving on my supervisory committee.

Finally, thank you to Dr. Viacheslav Adamchuk for his literally ground-breaking research in proximal soil sensor technology, and especially for his confidence and patience as my thesis supervisor. His exemplary work ethic and consistent commitment to his students foster a collegial, positive, and hard-working atmosphere in the PASS research team. I am sincerely grateful for his unfaltering support, both academic and financial, which made this thesis research possible.

Contributions of the Author

The DeepSoil framework brings together a variety of technologies. Many were created by the author, while others were provided by collaborators and third parties. The following are original contributions of the author:

- development of the jFocus custom microscope control software with autofocus routine for the Dino-Lite Edge 3.0 digital microscope in C++;
- drafting of the image acquisition protocol for the capture of high-quality soil-sample images (SSIs) using the microscope;
- manual review of SSI quality and annotation of suspected deficiencies;
- compilation of the AL2019 dataset comprising the SSIs and associated metadata including reference particle size distribution (PSD) measurements;
- development of Python code to systematically preprocess and transform SSIs and metadata from the AL2019 dataset for use with the PyTorch numerical computing library, including on-the-fly multiprocessing and disk caching via the Dask library;
- development of PyTorch-based code to define, train, and test CNN models for classification and regression tasks using parameter values configurable via file-based presets and/or command-line options;
- testing of known and custom CNN model architectures and identification of those most capable of being trained to predict soil PSD values from SSIs; and
- statistical analysis of soil PSD prediction performance of CNN models relative to baseline mean PSD-value predictions.

Table of Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface 28, 65
ASTM	American Society For Testing And Materials 2
CAD	Computer-aided Design iv
CCD	Charge-coupled Device 7, 10
CMOS	Complementary Metal-oxide Semiconductor 7
CNN	Convolutional Neural Network i, vi, xix, 8–14, 16–22, 24, 31, 36, 37, 40, 41, 44–49, 60, 63, 65, 66
CPU	Central Processing Unit xviii, 40, 41, 63, 85, 86
CRF	Conditional Random Field 8
CSSC	Canada Soil Survey Committee 2
CUDA	Compute Unified Device Architecture 64
DL	Deep Learning i, iv, xviii, 8, 11, 12, 14–16, 19, 64
EC	Electrical Conductivity 1
EV	Exposure Value 29, 30
FAO	Food And Agriculture Organization iii
FFT	Fast Fourier Transform 8, 10, 44, 46
FTP	File Transfer Protocol iv, 31, 35, 84–86
FV	Fisher Vector 18, 60
GMM	Gaussian Mixture Model 18
GMT	Greenwich Mean Time 30
GPU	Graphics Processing Unit xviii, 40, 41, 45, 46, 52, 63, 64, 85
HDR	High Dynamic Range 25, 27, 29

HMP	Hexametaphosphate 5
HWC	Height-width-channel 38
IR	Infrared 6, 12, 19
ISO	International Organization For Standardization 29
ISPA	International Society Of Precision Agriculture 1
ISSS	International Soil Science Society 2
JSON	JavaScript Object Notation 65
LED	Light-emitting Diode 25, 29
MAE	Mean Absolute Error 43, 54
MIR	Mid-infrared 6
ML	Machine Learning 8, 20, 36, 37, 47, 58, 64
MNIST	Modified National Institute Of Standards And Technology 14
MRF	Markov Random Field iv, 8
MSE	Mean-squared Error 43, 44, 54
MSG	Mean Squared Gradient 28, 29
NAS	Network-attached Storage 86
NSERC	National Science And Engineering Research Council iv
OM	Organic Matter 1, 5, 31–33
OS	Operating System xviii, 26, 27, 85, 86
PA	Precision Agriculture iii
PASS	Precision Agriculture And Sensor Systems iv, v, 25, 31, 35, 85, 86
PNG	Portable Network Graphics 30, 31, 36, 82
PSA	Particle Size Analysis xix, 1, 4, 6, 19, 31, 66
PSD	Particle Size Distribution vi, xvii, xix, 1, 2, 4, 7, 10, 19, 21, 45, 61
ReLU	Rectified Linear Unit 44
REST	Representational State Transfer 65

RGB	Red-Green-Blue 7
RMSE	Root-mean-squared Error 44
RNN	Recurrent Neural Network 11
S3	Simple Storage Service 35, 85, 86
SDK	Software Development Kit 27–29, 81
SGD	Stochastic Gradient Descent 15, 46, 47
SSI	Soil-sample Image vi, xvii, 20, 21, 30, 31, 34, 36, 49, 56, 65
USB	Universal Serial Bus 25, 26, 79, 80, 85
USDA	United States Department Of Agriculture 2, 3, 33, 84
VGG	Visual Geometry Group 16
Vis-NIR	Visible Near-infrared 6
YAML	YAML Ain't Markup Language 36, 49, 90

Glossary

A&L Canada	Partner commercial soil analysis laboratory located in London, Ontario, which carried out both capture of SSIs using the digital microscope and measurement of reference soil PSD values using the conventional hydrometer method. (`https://alcanada.com`) iv, 20, 31–33, 58, 85, 86
Albumentations	Python image augmentation/transformation library. (`https://albumentations.ai`) 38
Dask	Python library for efficiently loading and operating on objects, such as NumPy arrays, too large to fit in memory. (`https://dask.org`) vi
DinoCapture	Proprietary Dino-Lite microscope control software. (`https://www.dinolite.us/dinocapture/`) 26, 27, 81
Docker	Container virtualization technology enabling rapid spin-up of customized virtual environments. (`https://www.docker.com`) xvii, 87
effective diameter	The minimum diameter of a particle, i.e. the smallest opening through which it may pass. 2, 4
Git	Distributed version control system. (`https://git-scm.com`) xvii
GitLab	Cloud service for hosting Git software repositories and automating testing and deployment of software projects via Docker containers. (`https://about.gitlab.com`) 35, 85, 86, 89

jFocus	Custom-developed microscope control software. vi, 27, 29, 30, 32, 35, 81, 85, 86, 89
Jupyter	Web-based interactive Python shell. (`https://jupyter.org`) 49, 51
NumPy	Numerical Python computing library. (`https://numpy.org`) iv, xvii, 41
OpenCV	Computer vision and machine learning library. (`https://opencv.org`) iv
Pandas	Python data science library. (`https://pandas.pydata.org`) 35
python-ternary	Python library for generating ternary plots. (`https://github.com/marcharper/python-ternary`) 34
PyTorch	Python numerical computing framework which runs interchangeably on CPU or GPU, popular for implementing deep learning models. (`https://pytorch.org`) vi, 16, 41, 46, 61
virtual environment	Software-based platform that gives, to applications running within it, the appearance of a hardware, operating system (OS), and software context, independent of the actual underlying implementation. xvii, 87
Wasabi	S3-compatible cloud storage service. (`https://wasabi.com`) 35, 85, 86

Outline

The thesis presented herein is organized as follows: Chapter 1 provides a general introduction to the topics of soil particle size analysis (PSA) and computer vision. Chapter 2 presents a review of the literature in computer vision and deep learning as they relate to soil texture analysis. Chapter 3 describes, in detail, the materials and methods employed to carry out the research, including development of a digital microscope-based image acquisition system, compilation of a novel dataset of magnified images of soil samples labeled with reference particle size distribution (PSD) values, and training and testing of CNN regression models to predict those values based on the images. Chapter 4 presents the results of this research along with a statistical analysis and discussion thereof. In Chapter 5, avenues for future research are proposed. Conclusions are provided in Chapter 6, followed by references and appendices to complete the thesis.

Chapter 1. Introduction

At its heart, this is a thesis about dirty pictures – not the racy kind, of course, but literal pictures of dirt (*i.e. soil samples*) and the power they have to relay information about the contents of the soil from which they were drawn. Taking advantage of the decreasing cost of digital imaging and recent advances in the fields of computer vision and deep learning, the DeepSoil framework presented herein is a precision agriculture technology which aspires to harness this information to enable rapid, inexpensive, and therefore widely-available soil particle size analysis (PSA).

1.1 Precision Agriculture

The International Society of Precision Agriculture (ISPA) defines precision agriculture as "a management strategy that gathers, processes and analyzes temporal, spatial and individual data and combines it with other information to support management decisions according to estimated variability for improved resource use efficiency, productivity, quality, profitability and sustainability of agricultural production." Its goals include improved crop yields, reduced input usage (*e.g.* fertilizers and water), increased supply-chain efficiency, lessened environmental impact, and greater food security [2].

Proximal sensors for soil analysis are an active area of research within precision agriculture. Many types of sensors have been developed to measure a large variety of soil properties, such as chemical composition, acidity (pH), electrical conductivity (EC), organic matter (OM) content, gamma radiation [3], and particle size distribution (PSD). Soil analysis may be carried out either *in situ* [4], [5] or by collecting samples and transporting them to a specialized laboratory. Sensor fusion allows multiple soil properties to be measured simultaneously [6].

Accurate measurement of soil properties facilitates land-use planning by agricultural producers and regulatory bodies. For example, soil texture and organic matter content are important factors in the determination of appropriate crop selection, field management, and environmental practices [7].

1.2 Soil Texture

The particle size distribution (PSD) or *texture* of soil refers to its composition in terms of the relative mass of its particles with respect to effective diameter. Soil texture exerts a strong influence over physical, chemical, and biological soil properties, such as soil structure, aggregation, compaction, erosion, permeability, aeration, water retention, nutrient flow, root penetration, and microbial activity [8]. For example, whereas a soil predominantly composed of small (clay) particles will tend to be dense and relatively impermeable to air and water, a soil mostly composed of large (sand) particles will compact less and offer better aeration and drainage. *Loamy* soils contain a balance of particles at different scales and are generally favoured in the agricultural context because they foster structural, aerobic and hydrological conditions strongly supportive of root growth and soil ecology [8].

Soil texture classification systems assign names to specific ranges of soil particle dimensions, or *soil separates*. Several such taxonomies exist, including the USDA, CSSC, ISSS, and ASTM systems (Appendix A). For this thesis research, the United States Department of Agriculture (USDA) classification system was adopted. It divides soil particles into four broad soil separates according to their effective diameter: those smaller than $2\mu m$ are called *clay*, those ranging from $2\mu m$ to $50\mu m$ are called *silt*, those ranging from $50\mu m$ to $2mm$ are called *sand*, and all larger particles are called *gravel*. In the agricultural context, gravel particles are of limited interest, and USDA soil texture refers to the relative proportion (by mass) of particles in a soil belonging to each of the sand, silt, and clay separates.

Based on these proportions of soil separates, the USDA defines twelve soil texture classes: sand, silt, clay, loam, sandy clay, silty clay, sandy loam, clay loam, silty clay loam, sandy loam, silt loam, or loamy sand. A *loam* refers to a soil with balanced proportions of sand, silt, and clay particles.

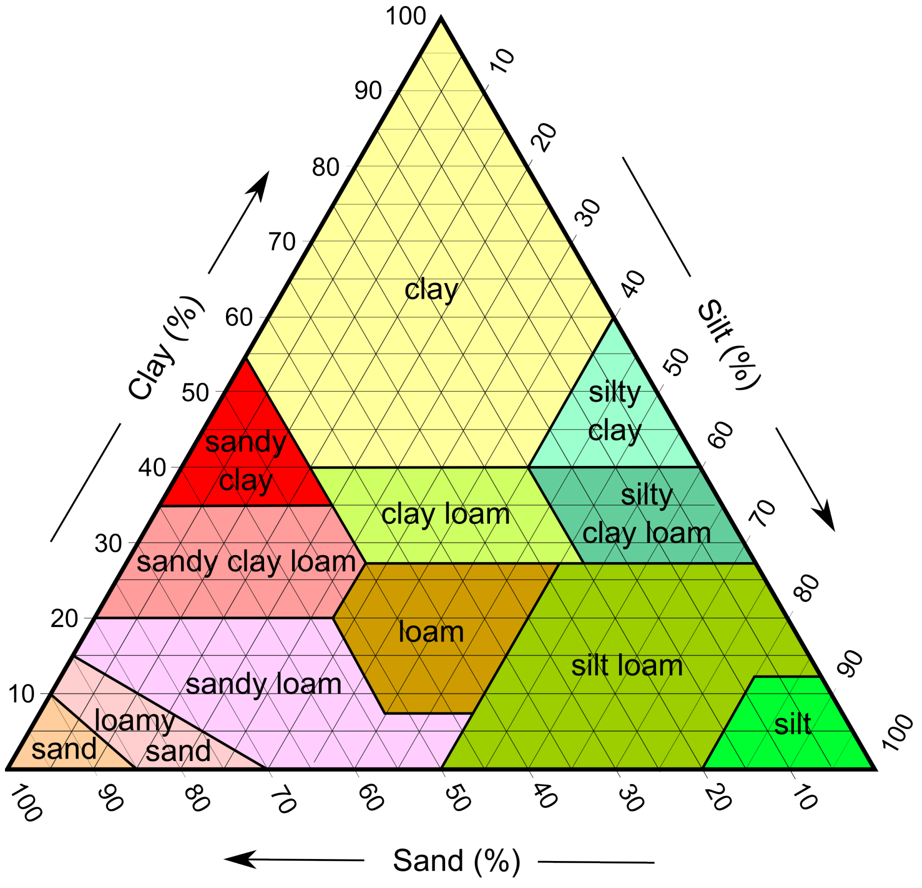


Figure 1.1: USDA Soil Texture Classes

It can be useful to visually display soil texture values with the help of a ternary plot such as that shown in Figure 1.1, which depicts the USDA soil texture classes as regions of different colors. A ternary plot is a representation of 3-tuples of values which invariably sum to a fixed value. In the case of USDA soil texture, the sum of the sand, silt, and clay portions is necessarily 100%:

$$Sand\ (\%) + Silt\ (\%) + Clay\ (\%) = 100\% \tag{1.1}$$

Each location within the texture triangle represents a potential soil texture value, and the sand, silt, and clay content of that value may be read off the graph by following the overlayed isolines to each one of the three axes. Notably, the isolines are not perpendicular to the axes, because a ternary plot is actually a projection of three-dimensional values onto the two-dimensional plane defined by the ternary constraint defined by Equation 1.1. Equivalently, it may be said that soil textures have only two *degrees of freedom*: as soon as two of the three sand, silt, and clay percentage values are fixed, the third is also fully determined.

1.3 Particle Size Analysis

Given the significant influence of particle size distribution over soil properties, its accurate measurement is of considerable importance both in soil science and in agriculture. Several methods of particle size analysis (PSA) have consequently been developed, from mechanical and sedimentation methods to newer methods employing electronic sensors.

1.3.1 Sieving

Next to getting a feel for a soil's texture by manipulating it between one's fingers, sieving is probably the most straight-forward method of particle size analysis. It proceeds in two steps: first, the soil sample is passed through a series of sieves with progressively smaller openings, such that particles of a given *effective diameter* are captured by the first sieve whose openings are too small to let them pass; then, the net weight of each sieving tray with and without its collected soil particles is measured to obtain the mass of particles within each diameter range [10]. Because the elongated dimensions of non-spherical particles may be larger than sieve openings, a shaker may be employed to help the particles reorient and find their way through (Figure 1.2).

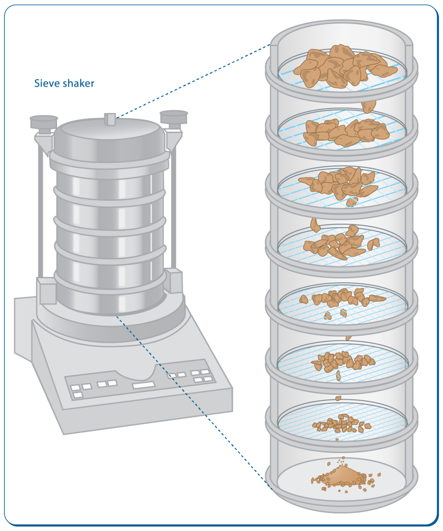


Figure 1.2: Sieving Method [9]

1.3.2 Sedimentation Methods

Sedimentation methods are predicated on the relationship between particle size and settlement rate in a liquid [10]. After pre-treatment to remove non-soil particles (*i.e.* organic matter, iron oxide, carbonate, and soluble salts) and dispersion of the soil particles, each soil sample is uniformly mixed with distilled water or a sodium hexametaphosphate (HMP) solution. Subject to certain assumptions [10] and the gravitational constant g, the sedimentation velocity v with which a particle of diameter d and density ρ_s will settle in a liquid of density ρ_ℓ and dynamic viscosity η is given by Stokes' Law:

$$v = \frac{g(\rho_s - \rho_\ell)d^2}{18\eta} \tag{1.2}$$

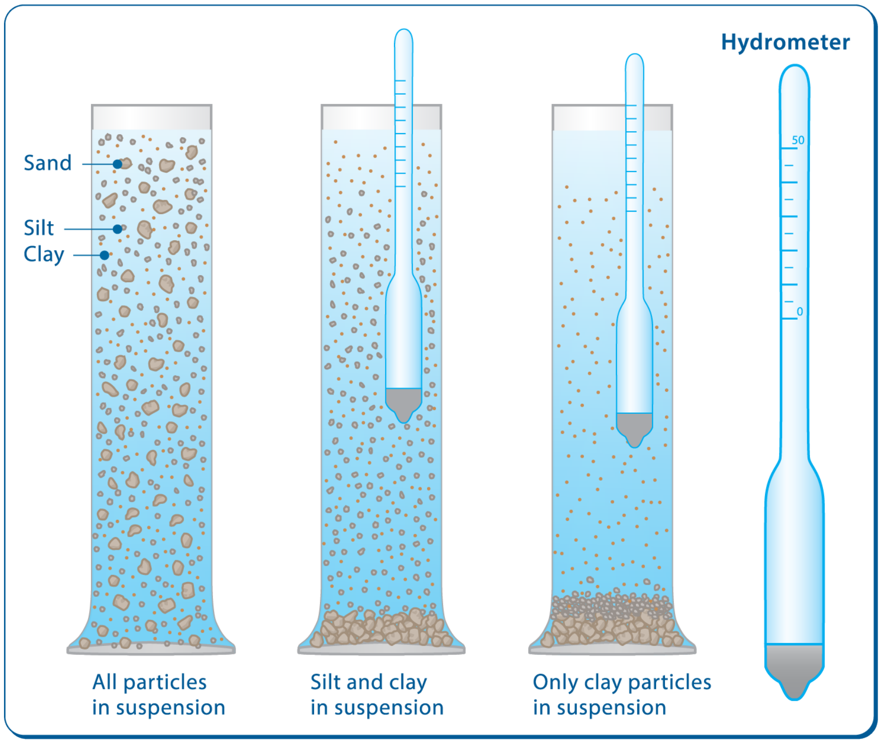

Hydrometer
Sand
Silt
Clay
50
0
All particles in suspension
Silt and clay in suspension
Only clay particles in suspension

Figure 1.3: Particle-Size Analysis by the Hydrometer Method [9]

The problem of measuring the particle size distribution of a soil sample then becomes one of measuring the variation of the density of the liquid soil suspension over time. This is typically performed using one of two standard approaches: the pipette method or the hydrometer method. In the pipette method, a set volume of the liquid suspension is directly sampled at a given depth, then dried and weighed. By Stokes' Law, the size of the particles in the sample reliably relates to the time at which the sample was taken, allowing the distribution of particle sizes to be calculated from the relative weights of the samples drawn at each time [11]. Alternatively, a hydrometer may be used to measure the relative density of the liquid soil suspension at regular intervals [12], [13]. As particles drop out of the suspension, the density of the fluid declines over time, with larger particles falling more rapidly than smaller ones (Figure 1.3). Again, according to Stokes' Law, the variations in the density of the liquid over time are quantitatively attributable to particles of given diameters. Charting the variation of relative density of the liquid soil suspension thereby yields the particle size distribution.

Sedimentation methods are quite accurate and therefore serve as the reference by which other methods are assessed [10]. However, they also require careful preparation of the liquid soil suspension and the ongoing attention of a soil laboratory technician to take regular measurements over the course of hours. The instruments themselves are not expensive, but the time and effort required to use them are a disadvantage when compared to modern electronic methods.

1.3.3 Electronic Methods

Electronic alternatives to sedimentation methods seek to do away with their meticulous wet chemistry, specialized laboratories, and labour. Instead, they promise rapid soil particle size analysis using electronic sensors of different kinds.

Laser diffraction particle size analyzers observe the spatial variations in scattered laser light interacting with a soil sample to measure texture [14]. While very accurate, these instruments are also very expensive [9], [15].

Infrared (IR) spectroscopy methods also have been applied to soil texture analysis [16]. Some rely on instruments that sense in the visible near-infrared (Vis-NIR) frequency band [5], [17]–[19] while others target the lower-frequency mid-infrared (MIR) band [20]. Soil texture values are calculated from the soil's IR response based on correlations identified using statistical and machine-learning techniques [21]. Infrared spectroscopy methods provide accurate measurements of soil PSD, but the instruments on which they rely are expensive, and they must be calibrated and regularly maintained [22].

Gamma radiation sensors assess soil texture indirectly by first measuring the intensity of high-frequency Gamma rays emitted from soil in the field, then correlating patterns in the radiation to soil composition [3]. While rapid and convenient, these estimates are not as accurate as measurements taken by dedicated instruments, and gamma ray sensors remain expensive relative to digital cameras and microscopes.

1.4 Computer Vision

Computer vision refers to a broad range of technologies for processing and analyzing digital images captured by cameras to extract and interpret the information they represent.

1.4.1 Digital Images

Conceptually, a digital image may be understood as a matrix of pixel intensity values $I(x, y)$ in at least two dimensions x and y, the magnitudes of which are defined as the image's width and height, respectively. In a typical grayscale image, eight bits are used to encode the intensity of each pixel, resulting in $2^8 = 256$ potential values ranging from 0 to 255. An intensity value of 0 represents a fully black pixel, a value of 255 represents a fully white pixel, and all intermediate values lie on the gray spectrum in between. A color image may be represented via the addition of a third dimension, the *color channel*, which results in a three-dimensional representation $I(x, y, c)$. Typically, there are three such channels, one for each of the RGB colors: red, green, and blue. The color intensity at each pixel location (x, y) is therefore represented as a 3-tuple of 8-bit RGB values, and combinations of these values represent a nearly universal range of colors.

A color digital camera couples optical components (*i.e.* lenses and mirrors) and an analog-to-digital sensor (*e.g.* CCD or CMOS) to produce digital images. The optical components direct and focus light collected via the aperture of the camera lens onto the sensor, which detects and translates the light intensity recorded in each of the frequency ranges corresponding to the red, green, and blue colors at each of its (x, y) pixel locations. The *resolution* of the camera denotes the total number ($W \times H$) of such pixels.

1.4.2 Deterministic vs. Adaptive Algorithms

Computer vision algorithms carry out a series of operations on digital images in order to transform and/or interpret their contents. These algorithms may be divided into two broad categories according to the type of strategy they employ. *Deterministic* algorithms process input images according to a fixed set of steps to systematically produce consistent output values without regard to past stimulus. Examples include edge-detection algorithms, various types of interpolation, blurring, affine transforms, and spectral transforms such as the Fast Fourier Transform (FFT) and wavelet transforms. These algorithms are typically fast, predictable, and do not require training to be effective, but they also depend entirely on the foresight and ingenuity of their authors to solve the problems with which they are presented. In contrast, *adaptive* algorithms include parameters which are adjusted in response to the inputs to which they are exposed, allowing them to internalize the effect of past stimulus and thereby learn from experience. These algorithms include probabilistic algorithms such as Markov random fields (MRFs) and conditional random fields (CRFs), as well as deep learning models.

Deep learning (DL), a specialization within machine learning, refers to a class of adaptive models characterized by a greater number of learnable parameters than those of simpler adaptive algorithms. In recent years, many DL models have been shown to outperform even the most rigorous traditional computer vision approaches at many tasks. One type of deep neural network in particular, the convolutional neural network (CNN), has proven especially effective at performing vision tasks such as classifying images according to their content [23]. For example, the very competitive ImageNet Challenge has been dominated by CNN techniques every year since 2012, when the image classification accuracy of the ground-breaking AlexNet CNN model leapt past that of all other approaches [24].

1.5 Research Objective

Given the success of convolutional neural networks (CNNs) models with respect to other vision tasks, the objective of this thesis was to investigate their suitability for particle size analysis, with the broader goal of facilitating rapid, inexpensive, and widely available soil texture assessment. More specifically, the aim was to develop, as a proof of concept, a deep-learning system capable of providing useful estimates of soil texture based on images of soil samples captured with an inexpensive digital microscope, using one or more CNN models trained to correlate learned features of those images to reference texture values measured using the hydrometer method.

Chapter 2. Literature Review

2.1 Image-Based Soil Particle Size Analysis

Image-based methods of soil texture analysis in the literature include spectral methods, a bag of visual words (BoVW) model based on Gabor filter banks, and a smartphone-based approach using random forest and CNN models.

Spectral methods convert the spatial information of pixel variations in images into the frequency domain. These approaches are motivated by the prospect that soil particles of given dimensions will demonstrate repeating spatial patterns of intensity in space, such that peaks and troughs in the frequency response of soil images will correspond to the soil's particle size distribution. Hryciw [25] applied the Fast Fourier Transform (FFT) to images captured with a CCD video camera suspended above soil samples. Few soil samples were analyzed and many images (video frames) were needed to analyze each sample. Breul [26] applied spectral methods to image texture and remarked on correlations with particle size distribution as between finer or coarser textures, but did not produce soil texture predictions from the images. Sudarsan et al. [22] measured the power spectrum of a continuous wavelet transform across frequencies to predict soil particle sizes belonging to two categories: coarse or fine. Prediction accuracy suffered when soil samples contained aggregates – clumps of small particles masquerading as larger ones.

Qi [27] used a bag of visual words (BoVW) model and multivariate partial least squares regression (PLSR) to quantify soil texture from digital microscope images. Visual features extracted from images included color and image texture. Coefficient of determination (R^2) of predicted sand, silt, and clay portions were 0.77, 0.68, and 0.71 and root mean squared error (RMSE) were 5.92%, 6.01%, and 2.98%.

Swetha [28] describes a method for predicting soil texture from images captured using a smartphone camera and analyzed using random forest and CNN models. The reported coefficient of prediction was very high for sand (0.97-0.98) and clay (0.96-0.98) and lower for silt (0.62-0.75). Only 90 soil samples were included in the study. Surprisingly, color features reportedly outperformed all other image-extracted features.

2.2 Image Texture of Soil Images

Other studies have analyzed microscope images of soil images with respect to image texture. Moran [29] describes a broad range of textural image features that appear in images of soil. Sofou [30] applied image segmentation algorithms to identify and segment regions of microscope images of soil particles according to their textural image features, with the aim of enabling the study of soil micromorphology. Marcelino [31] studied texture in microscope images to characterize soil sample microporosity.

However, it is important to distinguish *image texture* from *soil texture*: the latter refers to particle size distribution, while the former refers to the spatial arrangement of pixel intensities in an image, irrespective of the physical subject matter depicted. The above-cited studies were focused not on soil texture, but on image texture (albeit of soil images). As such, they did not propose methods to assess soil particle size distribution.

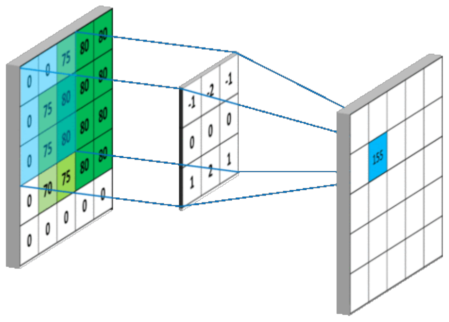
2.3 Deep Learning

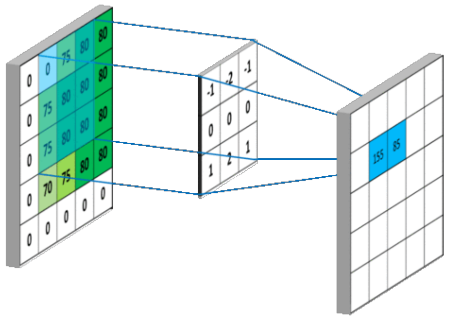
Deep learning (DL) refers broadly to a class of computational models consisting of multiple processing layers which may be adapted to learn abstract and often complex representations of data [32]. Different classes of deep learning (DL) models have been developed to tackle different types of problems, and many have proven disruptive in their ability to outperform even the most rigorous traditional approaches. For analysis of sequential signals such as speech or text, recurrent neural networks (RNNs) are favoured thanks to their ability to represent and accumulate contextual information over time [33]. For computer vision tasks, convolutional neural networks (CNNs) have become the *de facto* models of choice because they have proven very effective at representing, transforming, and analyzing the spatial information contained in images [23].

Deep learning models have been applied in a wide range of fields, including natural language processing [34], medical imaging [35], computational chemistry [36], computational biology [37], audio signal processing [38], image classification [23], semantic segmentation [39], and analysis of digital microscope images [40], just to name a few. In agriculture [41], CNN models have been trained to perform tasks such as plant recognition [42], fruit counting [43], crop classification [44], weed detection [45], and soil-type mapping from both remote-sensed images [46] and IR spectroscopy data [47].

2.3.1 Convolution Filters

The basic building block of all CNNs is the convolution operation, which takes the weighted sum of adjacent values from each region of an input tensor to produce output values as a new tensor. The *response* of the filter is given by the magnitude and arrangement of those output values. For example, a simple horizontal edge detection operation may be performed by convoluting a two-dimensional tensor (*i.e.* image) with a Sobel filter kernel [49] as shown in Figure 2.1. In the first frame (a), the values of a first region of a two-dimensional 5×5 tensor of pixel values (left) are multiplied by those of the Sobel filter kernel (centre) before being summed to generate a first output value in an output tensor (right). The filter kernel is then shifted to the right by one pixel, and the operation is repeated with respect to the pixel values of the resulting second input region (b). By repeating this operation for all valid regions, the complete response of the filter over the entire input tensor is computed (c). In this


(a) Sobel filter applied to a first region.


(b) And again to a second region.

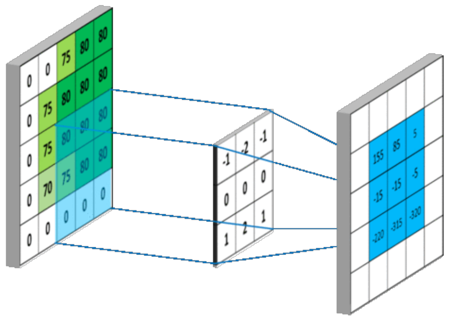

(c) And so forth to all valid regions.

Figure 2.1: Convolution Example [48]

example, the regions deemed *valid* were only those which allowed for complete overlap of the filter kernel with the input image, but partially-overlapping regions may also be of interest at times. To accommodate these cases, input values beyond the original input boundaries may be *padded* according to a variety of potential strategies, for example zero-padding or mirroring. The amount by which the kernel is shifted, known as the *stride*, may also be adjusted [50].

2.3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are typically composed of hundreds, if not thousands, of convolution filters, and early CNNs architectures such as the LeNet-5 model shown in Figure 2.2 are small by the standards of today [51]. Its architecture nonetheless illustrates the basic structure of CNNs as a series of stages each comprised of one or more *layers* of different types.

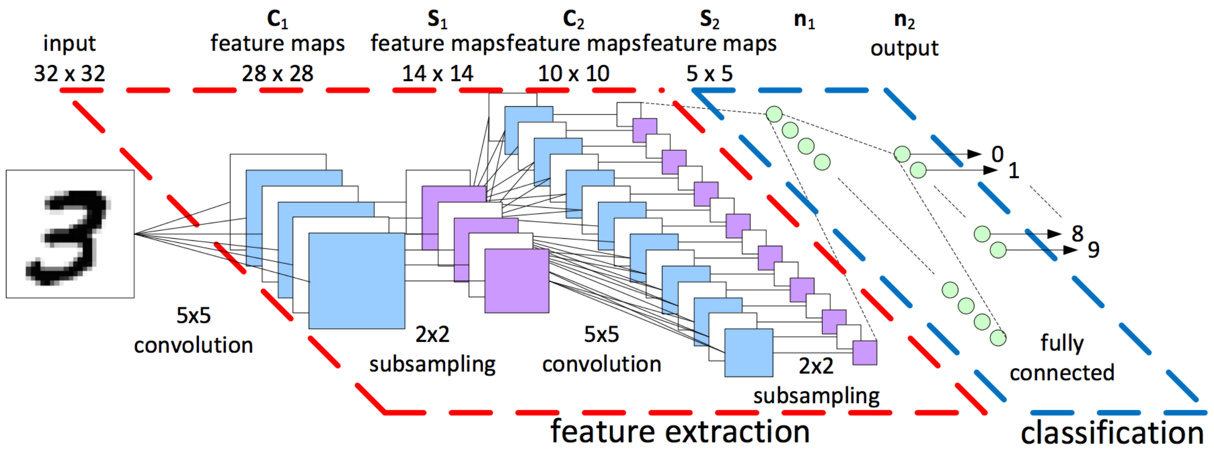


Figure 2.2: LeNet-5 CNN model for classifying handwritten digits. [51], [52]

An input image or other data is fed to the input layers of the network, and each layer generates a new representation of the data based on the representation of the previous layer. Long chains of layers encode meta-representations (*i.e.* representations of representations of representations of . . . of representations of data), which come to describe progressively more complex relationships in the input data as the depth of the network increases. Over time, multi-layered CNNs can be trained to discover and represent intricate structure in large datasets [32].

The initial feature extraction stage, composed of convolution layers and subsampling layers, aims to identify features of input images which are indicative of the target output value. Convolution layers are made up of banks of filter kernels similar to the Sobel filter kernel in the example presented above, with the important distinction that instead of having fixed weights at each pixel position, their weights are parameters which can be tuned over time, enabling the CNN model to learn from training data [50]. Subsampling or *pooling* layers may follow convolution layers to reduce the dimensions of the data flowing through the network and concentrate the learning process on image regions demonstrating the greatest filter response.

Because the goal of the LeNet-5 network of Figure 2.2 is to correctly classify MNIST handwritten digits from 0 to 9 [53], its feature extraction layers would likely identify shapes that visually distinguish the digits from one another. For example, the response of the convolution filters in the layers of the feature extraction stage would differ significantly when presented with a curvy '3' as compared to a very straight '1'.

The second stage of the network, identified as the classification stage in Figure 2.2, takes the output values of the feature extraction stage and combines them using a network of neurons to generate output values. Both the convolution layers of the feature extraction stage and the neurons of the *fully-connected* classification stage are governed by the values of their parameters. It is the optimization of these parameter values over time that constitutes the *learning* part of deep learning.

2.3.3 Parameter Optimization

Training a CNN model refers to the process of tuning the parameter weights of both its feature extraction layers and its output network layers through repeated exposure of the model to inputs and their corresponding labeled output values. Discrepancies between the values predicted by the model and the reference output values are measured according to a *loss function* and *backpropagated* through the network to tune its parameter values [54]. As a result, intricate structures in large data sets may be discovered [32].

Stochastic gradient descent (SGD) is the most popular optimization strategy for tuning model parameters [51]. The approach is heuristic rather than analytical because training optimal parameter values of complex neural networks is an intractable optimization problem. A model training strategy based on Stochastic gradient descent (SGD) operates by computing the gradient of a loss function $L(z_t, \theta)$ over the training data z_t based on the model parameters θ at the output of the network, then *backpropagating* the loss by the chain rule through the network to attribute responsibility for the loss to each of the model's nodes. The parameter values θ of the nodes are then iteratively updated by a small amount which can be tuned via a *hyper-parameter* ϵ_t known as the *learning rate* [55]:

$$\theta^{(t)} = \theta^{(t-1)} - \epsilon_t \frac{\partial L(z_t, \theta)}{\partial \theta} \tag{2.1}$$

The process is repeated over the entirety of the training data z_t for several epochs t until the loss function converges to a local minimum. A further hyper-parameter called *momentum* may be used to smooth the stochastic gradient and encourage faster training by averaging out noise in the gradient [56].

Alternatives to the SGD optimization strategy are also an active area of research. Two such alternatives are an adaptive gradient approach called Adam [57] and a modification thereof called AdamW [58]. A study critical of these approaches presented evidence that while their *training* performance was good, the generalization of the models they produced was poor – in other words, models were quickly trained to adopt parameter values that ultimately led to poor inference performance [59]. These concerns have been disputed in a recent study [60].

2.3.4 Explainability

One advantage of deep learning models over deterministic computer vision algorithms is that they learn from the data itself: no clever *a priori* design of filters is required. However, the seemingly magical ability of deep learning models to discover hidden patterns in data is also sometimes lamented as a weakness, because it can be difficult or impossible to understand how or why these models succeed at computer vision tasks when they do [61].

The lack of *explainability* of DL models is of particular concern when contemplating their use in contexts where health and safety are at risk [62]. In the realm of soil texture prediction, however, it is reasonable to sacrifice explainability for the sake of accuracy.

2.3.5 Survey of CNN Architectures

Deep learning is now a very active area of research, and new classes of CNNs are frequently emerging. New architectures typically center around novel network components designed to offer performance gains in terms of prediction accuracy, memory usage, and/or processing efficiency. A complete survey of this rapidly evolving field could be the subject of a thesis in its own right. Presented here is a selection of CNN architectures, chosen primarily out of convenience, these being available in the PyTorch library which was used to implement the present research.

LeNet

One of the oldest known CNN architectures, LeNet, was first proposed by Yann LeCun, widely regarded as one of the fathers of CNNs [54]. Despite being a very simple CNN by today's standards, it drew attention for its impressive ability to accurately classify handwritten digits [51].

Visual Geometry Group (VGG)

After the success of AlexNet at ImageNet 2012 [63], the emphasis of deep learning research turned to the investigation of ever larger and deeper network architectures. The VGG-16 (Figure 2.3) and VGG-19 CNN architectures developed by the University of Oxford's Visual Geometry Group (VGG) took the brute force approach of adding more layers, with each model being named for the number of parameterized layers it included [64]. The VGG-19 model was composed of 16 convolution layers and 3 fully-connected layers, along with the 5 deterministic MaxPool layers and 1 SoftMax layer. These large networks offered performance gains over smaller networks at the expense of longer training times and higher resource requirements.

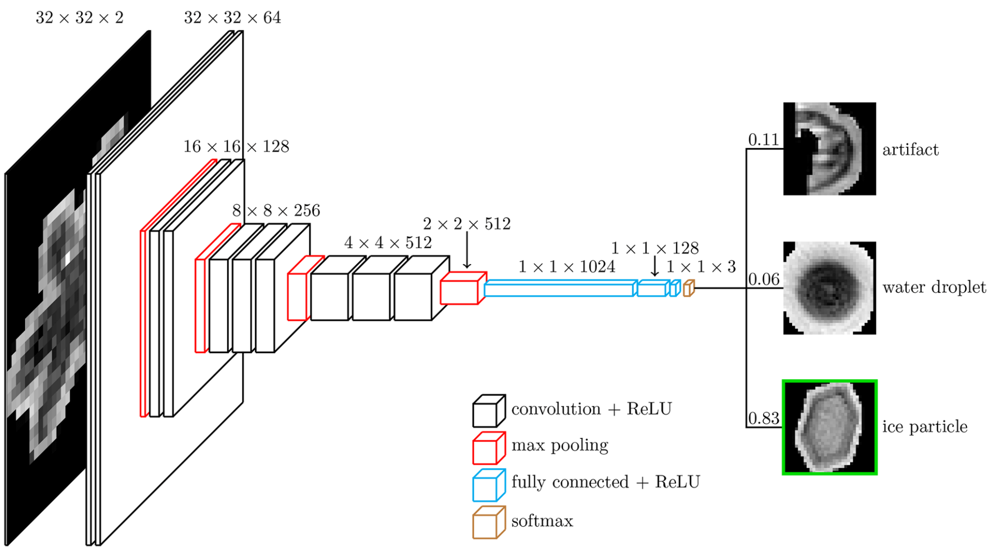


Figure 2.3: VGG-16 CNN model for classifying cloud particles as water or ice. [64], [65]

ResNet

Prior to the introduction of ResNets, the depth of CNNs was limited because of a problem known as the *vanishing gradient*: the tendency for gradients back-propagated to earlier layers of the network during optimization to become smaller and smaller until they effectively disappeared, thereby making it difficult to train deep networks [66]. ResNets overcame this problem through the use of *residual blocks* which included *identity shortcut connections* that have the effect of lessening the attenuation of the gradient during back-propagation [67]. As a result, ResNet architectures enable training of much deeper networks. Popular variants include the ResNet50 and ResNet101 networks with 50 layers and 101 layers, respectively.

MobileNetV2

Breakthroughs in CNN design, such as residual blocks, enabled training of deeper and more expressive networks. But as networks grew deeper, the number of parameters they used also tended to expand, placing higher demands on computing power and especially the amount of memory required to train them and apply them to inference tasks. In response, some researchers have focused on CNN architectures better suited to resource-constrained contexts. The MobileNet and MobileNetV2 architectures introduced *inverted residual structures* with shortcut connections between *bottleneck* layers. The bottleneck layers reduce the dimensionality of representations at various stages in the network, thereby substantially reducing the number of model parameters and their associated memory requirements [68].

EfficientNet

Another class of CNN models that has been attracting attention recently are so-called EfficientNets [69]. Like the MobileNetV2 architecture that ultimately proved most successful for this thesis research, EfficientNet CNN designs specifically target resource-constrained contexts, aiming to balance training and inference performance based on a processor and memory budget.

Fisher Vector CNNs

Fisher vectors are representations that store the mean and covariance of Gaussian mixture models (GMMs) of input values. Using these descriptive statistics of input representations rather than the representations themselves divorces their outputs from the spatial information contained in their inputs. In other words, whereas the input to a Fisher vector (FV) is spatially ordered, its output is an orderless representation of the input. Cimpoi [70] proposed and Andrearczyk [71] refined the use of FV layers in a specialized FV-CNN model architecture targeted to image texture analysis. Because image texture refers only to local features contained within regions of images, it may be counterproductive to propagate spatial information through all of the layers of the network. By introducing one or more Fisher vector layers, the responses of the preceding convolutional layers could be pooled without regard to their spatial information, such that textures represented anywhere in an input image would result in the same response. Moreover, these FV pooling layers, like bottleneck layers, reduced the dimensionality and therefore the memory footprint of representations, resulting in improved training and test performance while preserving image texture classification accuracy [70].

2.4 Summary

Several techniques for soil particle size analysis are known: sedimentation methods exploit the reliable relationship between particle size and fluid settlement rate given by Stokes' Law; electronic methods infer soil texture from observed patterns in the interaction of soil particles with electromagnetic radiation recorded at one or more wavelengths; and image-based methods correlate spatial patterns in light intensity to particle size distribution. While instances of each one of these techniques have been shown to effectively measure soil texture under the right conditions, none has proven ideal in all cases.

In general, a balance must be struck between speed, accuracy, and cost. Sedimentation methods are accurate but slow, because they require soil samples to be transported to a suitable laboratory, and costly, because they demand meticulous analysis by a specialized technician over the course of hours. Electronic methods are much faster, but they rely on instruments that are either expensive (*e.g.* laser and IR instruments) or of limited accuracy (*e.g.* cameras). Image-based approaches promise to augment the accuracy of inexpensive cameras using clever computer vision algorithms. A handful of studies have taken this approach, but none has applied CNNs to a large and varied dataset of microscope images.

While not a silver bullet, the proposed approach presents several potential advantages over known methods: digital microscopes cost less than laser diffraction or infrared spectroscopy instruments, image analysis is much faster than conventional wet chemistry methods, and CNNs are more adaptive than deterministic computer vision techniques, allowing their accuracy to improve over time as they learn from labeled soil-sample images. Moreover, given the rapid pace of ongoing research in deep learning, a CNN-based framework for prediction of soil texture, however modest, would be well-positioned to benefit from emerging breakthroughs.

Chapter 3. Materials and Methods

3.1 Overview

The overall approach of the DeepSoil project may be summarized in four steps:

1. In partnership with a commercial soil analysis laboratory, A&L Canada Laboratories, collect a large set of digital microscope images of soil samples and associated reference soil texture values measured using one or more of the standard methods (*e.g.* the hydrometer method).
2. Augment the set of digital images by applying non-destructive image transformations.
3. Train several convolutional neural networks (CNNs) to correlate features of the augmented soil-sample images (SSIs) to soil texture reference values according to well-known machine learning practices – segment the dataset into training and test sets, then iteratively adapt parameters to minimize a chosen loss function.
4. Evaluate the relative performance of the trained CNNs to identify the combinations of network structure and training hyper-parameters which yield the model with the greatest ability to predict soil texture from digital microscope images.

3.2 Logical Architecture

The DeepSoil framework may be understood as a logical architecture sitting on top of a number of networked physical devices, the logical architecture comprising a set of interrelated components defined by the role each plays in carrying out the overall goal of assessing soil texture from digital microscope images, and the physical devices being the particular hardware, software, and virtual (*i.e.* cloud) components which serve to implement the logical components. Specifications of the latter are provided in Appendix E whereas this chapter focuses on the former.

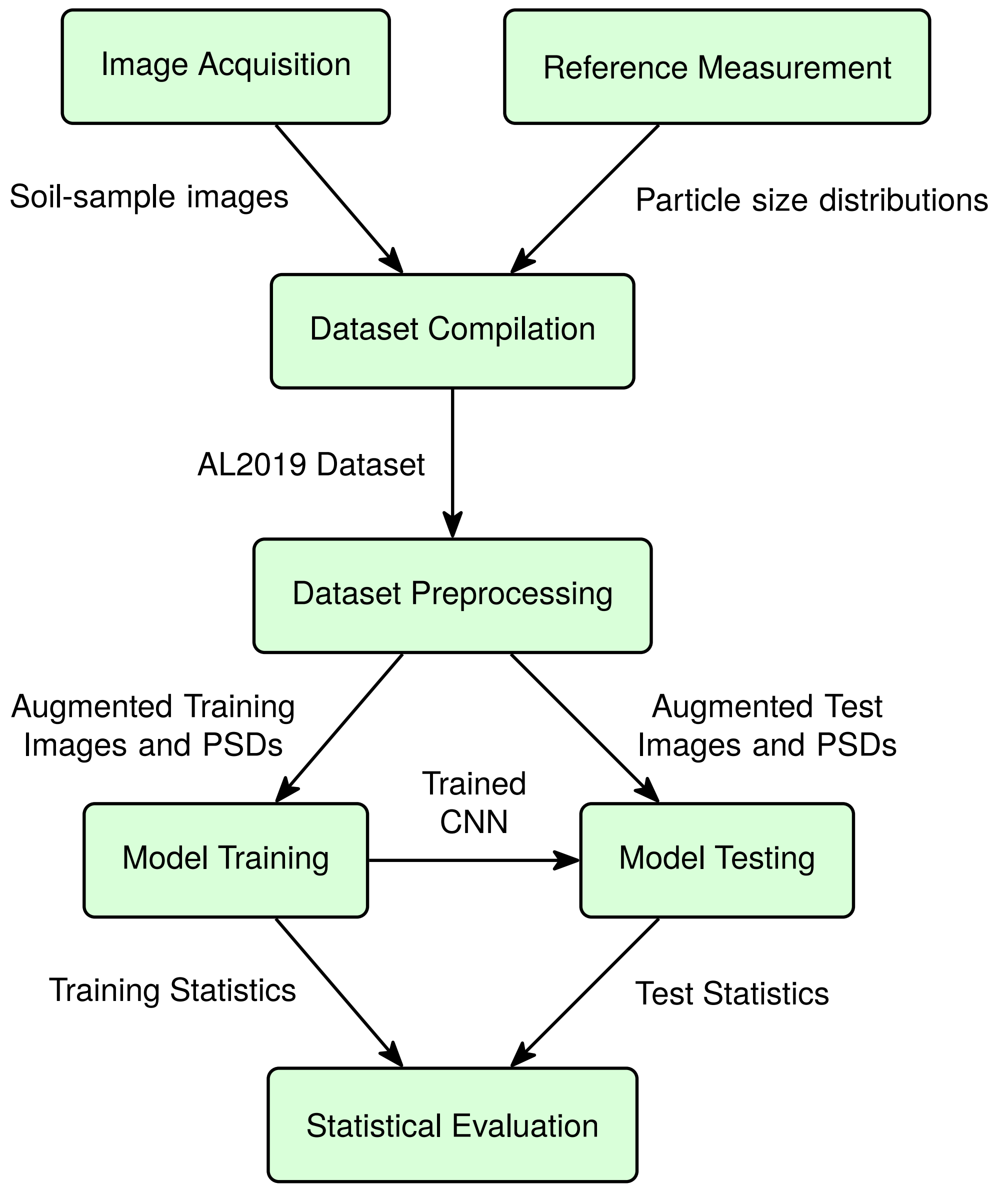


Figure 3.1: DeepSoil Logical Architecture

The overall logical architecture of the DeepSoil framework, shown in Figure 3.1, includes seven major components. First, an image acquisition component comprising a high-resolution digital microscope connected to a computer equipped with microscope control software. Second, a reference measurement component being a soil laboratory with experience measuring particle size distributions of soil samples using a conventional method known to be reliable (*e.g.* hydrometer or pipette), the measurements serving as target values to be hypothetically replicated on the basis of the images. Third, a dataset compilation component responsible for combining the acquired images with their respective reference texture measurements and reliably storing the combination as a comprehensive dataset in a computer-readable format well-suited to algorithmic analysis. Fourth, a dataset preprocessing pipeline to load, transform, split, and efficiently feed the images and metadata of the dataset to the fifth and sixth components, namely the model training component and model testing components, the former being responsible for heuristically adapting the parameters of a CNN model to minimize a loss function evaluated over a training set, and the latter being used to evaluate the predictive performance of the trained model over a test set, the training and test sets being mutually exclusive subsets of the complete dataset. Finally, a statistical evaluation component to collate and compare performance metrics of multiple training and test runs across a variety of CNN model architectures in order to identify those best suited to predicting target soil texture values from digital microscope images. Detailed descriptions of each of these components are provided in the following sections.

3.3 Image Acquisition System

3.3.1 Requirements

While it is sometimes possible for computer algorithms to recognize features in images that human observers cannot [72], it is generally imprudent to expect them to do so. For example, an object which is completely occluded in a scene is no more detectable by an algorithmic observer than a human one. Conversely, images with obvious features such as objects with regular geometric shapes of uniform color set against a background of a visually-distinct color make identification of those objects trivial for humans and computers alike. Most interesting computer vision tasks – such as identifying crosswalks or vehicles in a street scene, as netizens have become accustomed to doing in the era of Captcha[1] – lie somewhere in between.

The success of such non-trivial computer vision tasks often hinges on the quality of the images which serve as inputs to their algorithms. In this regard, image *quality* ought to be defined instrumentally with respect to a particular task – that is, images should be deemed to be of high quality to the extent that they render visible or highlight the features of a scene which provide information essential to that task's performance [73]. The requirements of an image acquisition system therefore follow from an inquiry as to which image features are contextually relevant.

What features of an image of a soil sample ought we reasonably expect to enable accurate assessment of the distribution of particle sizes present therein? The presumptive answer is any and all features that enable or enhance the visibility of the particles, such that their relative frequency may be assessed. More specifically, the components, software and procedures of the image acquisition were selected and/or developed with the following requirements in mind: focus, magnification, resolution, consistency, and reliable labeling.

Focus

Seeing individual particles *as individual particles* necessarily entails being able to distinguish among them – or more precisely, to detect the *boundaries* between them. Humans identify these boundaries by observing spatial variations in light intensity and color. More specifically, when looking closely at grains of sand with the naked eye, we distinguish the grains from one another by observing the nature and degree of these variations: where light intensity and/or color vary sharply, we see an *inter*-particle variation (*i.e.* a grain-grain boundary) and thereby identify the presence of two separate particles; however, where these properties vary slowly, we see an *intra*-particle variation: a continuous pattern belonging to a single, unbroken grain of sand. The ability to correctly assess spatial variability of light intensity and color is thus critical to human observation of soil sample particles. Barring contradictory evidence, we should likewise expect sharp focus to be of paramount importance in the context of soil particle size distribution analysis

[1] http://www.captcha.net

by computer vision algorithms. Therefore, a first requirement of the image acquisition system is that it ought to produce images of the best possible focal quality.

Magnification and Resolution

Concomitantly, to the greatest extent possible, the translation of the optical image into digital pixel values must be carried out at resolution great enough to render visible the sharp boundaries of particles at the scale of clay (diameter $< 2\mu m$). This requirement suggests that magnification levels should be set as high as possible and that images should be captured at the highest possible resolution. However, a countervailing consideration suggests that magnification levels must not be set too high, lest individual sand particles – whose diameters may be up to $2mm$, some $1000\times$ larger than those of clay particles – be rendered invisible by virtue of filling the entire image. Just as proverbial forests may fade into the background when we observe them at the scale of trees, so too might the boundaries between sand particles fail to be observed when the microscope is zoomed in too close. This consideration will be revisited in section 5.3.

Consistency of Image Capture Settings

Consistency is also important. Image capture settings and conditions should change as little as possible across all images, such that only the soil sample under analysis varies between captures, otherwise the CNN models may learn patterns in the visual artifacts resulting from varied settings instead of features of the soil samples themselves.

Reliable Cross-Referencing with Soil Samples

Finally, images must be uniquely identified and easily associated with their respective soil samples and reference measurements.

3.3.2 Digital Microscope

Acting as the sensor bridging the physical and digital worlds, the microscope selected for this project was the Dino-Lite Edge 3.0 `AM73915MZT` (Figure 3.2), manufactured by AnMo Electronics Corporation of Hsinchu, Taiwan [74]. Its most important features include its 5.0 megapixel resolution (2560×1920), 24-bit colour depth, $220\times$ optical magnification, built-in LED lighting to illuminate the sample under analysis, adjustable polarization filter, motorized lens, support for high dynamic range (HDR) imaging, and Universal Serial Bus (USB) 3.0 connectivity.

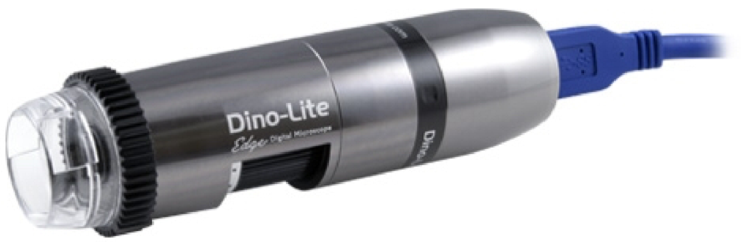


Figure 3.2: Dino-Lite Edge 3.0 `AM73915MZT` Microscope [74]

3.3.3 Microscope Holder

A custom 3D-printed solid plastic holder previously designed, manufactured, and patented by past and current members of the PASS research team was used to house the microscope [75]. Its primary functions were to protect and stabilize the microscope at an adjustable distance from the sample under observation while preserving access to the microscope's physical controls. As depicted in Figure 3.3, the microscope holder secured the microscope within its inner cavity by means of a manually-tightened plastic screw (not shown) and oriented the microscope upwards toward a flat, abrasion-resistant glass window on which a soil sample could be placed. The window itself was housed in a cap at the top of the holder, the cap being attached to the lower portion of the holder via threaded ridges, such that the focal distance from the lens of the microscope and the soil sample could be coarsely adjusted by manually rotating the cap. A first opening located along the column of the microscope holder allowed access to the microscope's magnification dial, and a second opening at the bottom of the holder (not shown) allowed access to its USB port for connection, via a USB cable, to a computer running microscope control software.

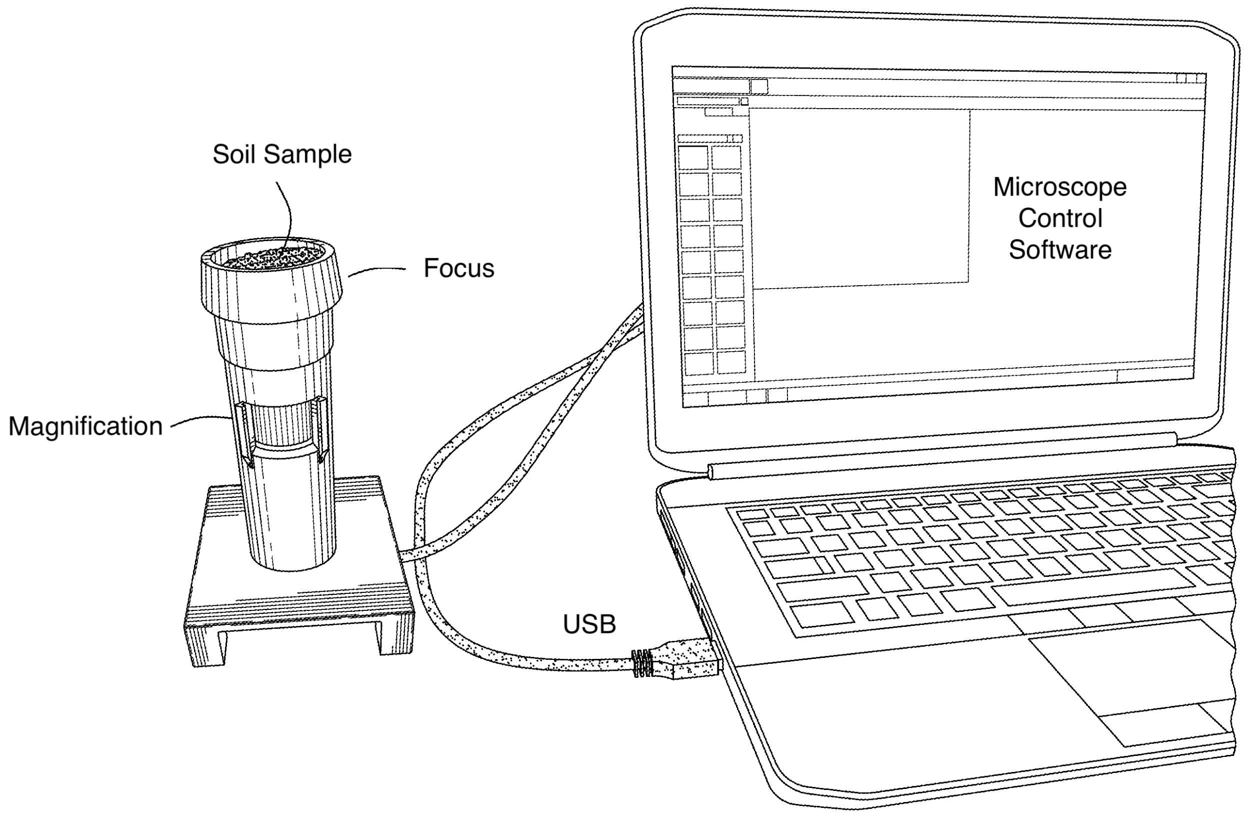


Figure 3.3: Image Acquisition System (adapted from [75])

3.3.4 Microscope Control Software

DinoCapture

Initial experiments with the Dino-Lite microscope were performed using the DinoCapture (version 2.0) microscope control software application for the Microsoft Windows OS, downloaded from the Dino-Lite website. The DinoCapture software provided several features essential for capturing images, such as the ability to control the camera's lights and capturing images in various file formats; however, some features were lacking. Most notably, the DinoCapture software was unable to automatically adjust the focal length of the microscope's lens using the built-in motor in order to achieve a consistently crisp focus. This meant that the focus on the microscope needed to be manually adjusted by its human operator, requiring both dexterity to rotate the microscope holder cap by minute angles and discriminating subjective visual analysis of the preview image. In practice, the technician at the partner laboratory reported repeated difficulties adjusting the focal length to obtain images of a satisfactory focus.

A second limitation of the DinoCapture software was its inability to automatically associate metadata with images as they were captured, increasing the risk of operator error in mislabeling or failing to label images of soil samples with the correct sample number.

jFocus

Given the paramount importance of obtaining sharp, correctly labeled images for this project, it was decided that specialized microscope control software should be developed. On its website, Dino-Lite offers a software development kit (SDK) to facilitate the development of custom software for its microscopes.[2] The SDK includes example source code in various programming languages for a simple program to control the microscope via an ActiveX control called `DNVideoX`. For this project, a custom microscope control software application, dubbed jFocus, was developed for the Windows 10 OS using the C++ version of the SDK by modifying the source code of this sample program. First, unnecessary bells and whistles were trimmed out, thereby reducing the risk of critical settings being deliberately or inadvertently altered between image captures. The result was a simple graphical interface with an uncluttered presentation and little opportunity for operator error. A number of new features were then added on the basis of the aforementioned requirements, the most important of which were a custom autofocus routine, activation of the microscope's HDR imaging capability, and hard-coding of essential settings for consistency across image captures.

[2]https://www.dinolite.us/en/sdk

Autofocus Routine

As discussed in section 3.3.1, one of the critical requirements of the image acquisition system was to maximize the visibility of the soil sample's particles by rendering spatial variations in the image as sharply as possible. In computer vision, the spatial variability of pixel intensity in an image is known as the image *gradient*. Mathematically, the gradient $g(x, y)$ of pixel intensity $I(x, y)$ at each location (x, y) of an image may be calculated by applying a horizontal Sobel filter to obtain the horizontal gradient $\frac{\partial I}{\partial x}$ and a vertical Sobel filter to obtain the vertical gradient $\frac{\partial I}{\partial y}$, then taking the root of the squares of the two filter responses according to Equation 3.1 [76]:

$$g(x,y) = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \tag{3.1}$$

When an image is in focus, objects in the image are sharp and the gradient at the boundaries between them is high. When the image is out of focus, however, light which ought to be concentrated within a single pixel is also diffused to adjacent pixels, such that pixel values bleed together and the boundaries between objects soften – *i.e.* gradients are reduced – while regions which ought to be of uniform intensity may see their gradient increase as adjacent colors bleed into them. Taken over the entire image, the average gradient may not be radically altered by poor focal quality. However, taking the square of the gradients at each pixel location further emphasizes large gradient values, such that the mean squared gradient (MSG) of an image of height H and width W, given by Equation 3.2, may serve as a metric of image focus [77]:

$$\textit{MSG} = \frac{1}{HW}\sum_{x,y} g(x,y)^2 \tag{3.2}$$

The sharpest focus is obtained at a precise focal distance between the microscope lens and the soil sample – the image will appear out of focus if that distance is either too great or too small. The Dino-Lite SDK's C++ application programming interface (API) included functions to control a small motor in the microscope to adjust the focal distance as well as a `SobelCenter` function to compute the gradient in a central region of the captured image.

With these functions as building blocks, an autofocus routine was added to the jFocus software, which first set the focal distance to one extreme and computed the MSG, then varied the focal distance in small increments until the maximal MSG value was reached.

High Dynamic Range (HDR) Photography

A conventional digital image is taken using a single exposure of the camera's light sensor array. In low light conditions, the camera's aperture size, exposure time, and/or ISO value (*i.e.* digital gain [79]) are increased, the cumulative effect of which is to increase the camera's exposure value (EV), meaning simply that the camera becomes more sensitive to variations in luminosity. However, if light levels are increased, the sensor of a camera set to a high EV will quickly become saturated, resulting in unvariegated white regions within the image. In such cases, the aperture size, shutter speed, and/or ISO value must be adjusted to reduce the EV, restoring the camera's light sensitivity to a lower level appropriate for capture of sub-maximum (saturated) pixel values.

In HDR photography, multiple exposures captured at different EVs are superimposed, enabling observation of a wider range of light levels within a single frame (Figure 3.4) [80]. The Dino-Lite SDK's `DNVideoX` ActiveX control allows such images to be captured via an Extended Dynamic Range mode which automatically varies both the quantity of light emitted by the microscope's light-emitting diodes (LEDs) and the microscope's EV. The jFocus software operated exclu-

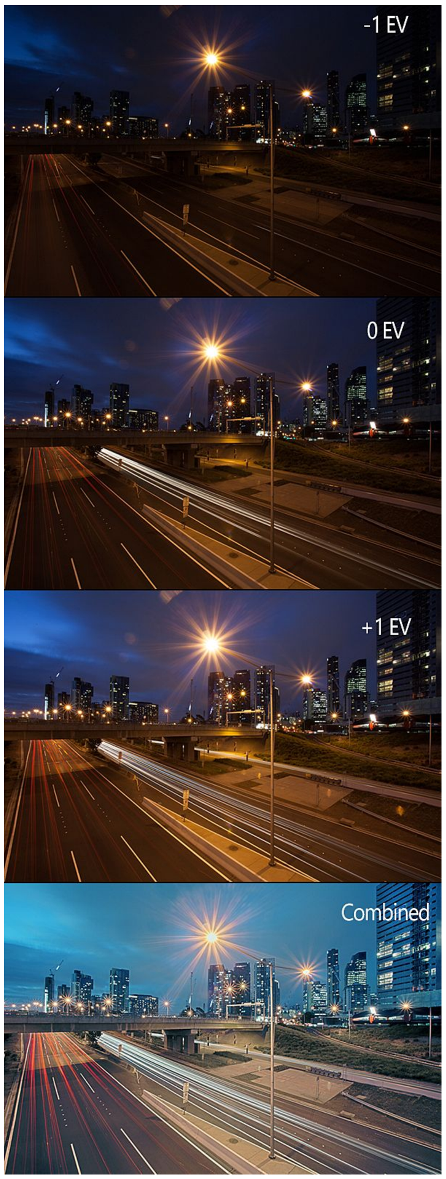


Figure 3.4: HDR compositing of multiple exposures. [78]

sively in this mode, thereby eliminating any bias that may be introduced by relying on operator judgment with respect to EV settings.

Other Hard-Coded Settings

Several further design choices were hard-coded into the jFocus software to ensure that consistent settings would be applied across all soil-sample image captures. The resolution and colour depth were set to the maximum values supported by the Dino-Lite microscope, namely 2560×1920 pixels in 24-bit colour. Images were saved in the Portable Network Graphics (PNG) file format and compressed using lossless compression to reduce their size on disk while preserving all original pixel values as recorded by the microscope. Essential metadata about the captured images were preserved in the filenames of image files, including the jFocus software version number, a GMT timestamp indicating the date and time of capture, and a mandatory user-entered sample identification number uniquely identifying the soil sample for cross-referencing with a table of reference values measured using the hydrometer method (see section 3.4).

3.3.5 Image Acquisition Protocol

Together with the microscope, holder, and jFocus software, the image acquisition system also included a detailed set of instructions explaining the required steps to be taken during initial setup and when capturing microscope images of soil samples. Important elements of the protocol included setting the magnification level to the microscope's maximum value (~220$\times$), activating the polarization filter to reduce glare from reflective soil particles which may otherwise cause saturation of the camera sensor [81, ch. 8], securing the microscope firmly within the holder to avoid motion blur, even spreading of 20 grams of soil from each sample on the microscope window, and cleaning of the window with a microfiber cloth between samples. The complete protocol as distributed to the partner laboratory which carried out the image acquisition process is included as Appendix B for reference.

3.3.6 Verification of Images

The work of capturing digital microscope SSIs and taking hydrometer measurements (section 3.4) was carried out in several batches by collaborators at the partner laboratory, A&L Canada, as new soil samples were received and processed from May to December of 2019. The batches of PNG images were regularly uploaded to an FTP server located in the PASS Lab at McGill University's Macdonald Campus. In total, 829 images and associated metadata were provided.

A manual review of the received SSIs was performed by visually appreciating the images and noting any deficiencies with respect to image quality. In particular, it was observed that, in some cases, the soil sample had not been adequately pressed up against the microscope holder window prior to image acquisition, such that regions of the affected images contained air gaps characterized by visible loss of focus. Of the 829 images received from the partner laboratory, 34 exhibited such deficiencies and were labeled as suspicious and systematically excluded from consideration, leaving a total of 795 verified images for testing and training of the DeepSoil CNN models.

Along with images of soil samples, the partner laboratory also sent digital microscope images of a calibration target with markings in increments of $0.1mm$. By measuring the distance in pixels between photographed markings, it was determined that the pixel density of the images was $1500pixels/mm$, or equivalently, that the horizontal and vertical dimension of each pixel was $6.67\mu m$. At that resolution, the $2\mu m$ clay-silt boundary corresponded to 3 pixels in diameter, and the $50\mu m$ silt-clay boundary was 75 pixels in diameter. The total area represented in each 2560×1920 image was $1.71mm \times 1.28mm$. Theoretically, larger sand particles up to the $2mm$ sand-gravel boundary were therefore clipped. A proposal to address this issue in future work is addressed in section 5.3.

3.3.7 Organic Matter Content and Four-way Texture

As described in section 1.3.2, organic matter (OM) is normally eliminated from soil samples when they are pretreated for particle size analysis using sedimentation methods such as the hydrometer method. However, perhaps for reasons of procedural convenience, pre-treatment of soil samples for OM removal was not performed on soil samples by the partner laboratory before capturing their images with the digital microscope, and sometimes substantial amounts of organic matter were present in the soil samples when the images were captured.

Consequently, it may be useful to consider the textural composition of the soil samples as a 4-tuple $(sand_4\ silt_4\ clay_4\ OM)$ in addition to the usual $(sand_3\ silt_3\ clay_3)$ 3-tuple of values. As a shorthand, the former will be termed *four-way* texture and the latter *three-way* texture in this dissertation. The subscript accompanying the textural component provides an indication to the number of texture dimensions. The four-way texture values may be obtained by normalizing the three-way values by the non-OM soil proportion as follows:

$$sand_4 = (1 - OM) \cdot sand_3 \tag{3.3}$$

$$silt_4 = (1 - OM) \cdot silt_3 \tag{3.4}$$

$$clay_4 = (1 - OM) \cdot clay_3 \tag{3.5}$$

3.4 Reference Laboratory Measurements

The partner laboratory, A&L Canada, is a commercial soil laboratory with expertise in measuring properties of soil samples for its clients across North America. In many cases, these properties include soil texture values measured using the conventional hydrometer method as described in section 1.3.2. Along with the images of soil samples taken using the image acquisition system described above, they provided soil texture values for those soil samples in a collection of Microsoft Excel spreadsheets, with entries identified using the same unique soil sample identification numbers that were entered into the jFocus software when capturing the images.

Component	Min	Max	Median	Mean	StdDev
$sand_3$	3.6%	98.3%	53.3%	54.9%	21.7%
$silt_3$	0.0%	71.9%	29.1%	28.1%	14.5%
$clay_3$	0.4%	73.4%	14.0%	17.0%	11.2%

Table 3.1: AL2019 Three-way Texture Distribution (795 Samples)

The statistical distribution of three-way soil texture values for the 795 soil samples is shown in Table 3.1. The measured OM proportion was provided by the partner laboratory for only 624 of the 795 soil samples; measurements were not available for the other 171 samples. Based on the OM data, the four-way texture values for the sand, silt, and clay components were calculated for those 624 samples using Equations 3.3 to 3.5. The distribution statistics for the resulting four-way textures are provided in Table 3.2.

Component	Min	Max	Median	Mean	StdDev
$sand_4$	5.2%	97.5%	55.9%	54.9%	22.1%
$silt_4$	0.0%	57.3%	24.8%	25.5%	14.3%
$clay_4$	0.4%	70.8%	12.8%	15.6%	10.7%
OM	0.1%	56.8%	3.0%	4.0%	4.9%

Table 3.2: AL2019 Four-way Texture Distribution (624 Samples)

Along with reference soil texture measurements, A&L Canada also provided the USDA soil texture class for each sample in a field named `STCLASS` (Table 3.3). Loam and sandy loam soils were most prevalent, while no silt samples and just one sandy clay sample were included. This distribution of soil textures reflects the range of soil samples received from the partner laboratory's clients for analysis in the course of its regular commercial operations. A visual representation of the distribution across USDA soil texture classes is shown in the ternary chart of Figure 3.5.

`STCLASS`	Class Name	Samples	Prevalence
1	Clay	30	3.8%
2	Silt	0	nil
3	Silt Loam	28	3.5%
4	Silty Clay Loam	18	2.3%
5	Clay Loam	66	8.3%
6	Sandy Clay	1	0.1%
7	Sandy Clay Loam	37	4.7%
8	Loam	207	26.0%
9	Sandy Loam	249	31.3%
10	Loamy Sand	113	14.2%
11	Sand	36	4.5%
12	Silty Clay	10	1.3%

Table 3.3: AL2019 Soil Sample Prevalence by USDA Texture Class

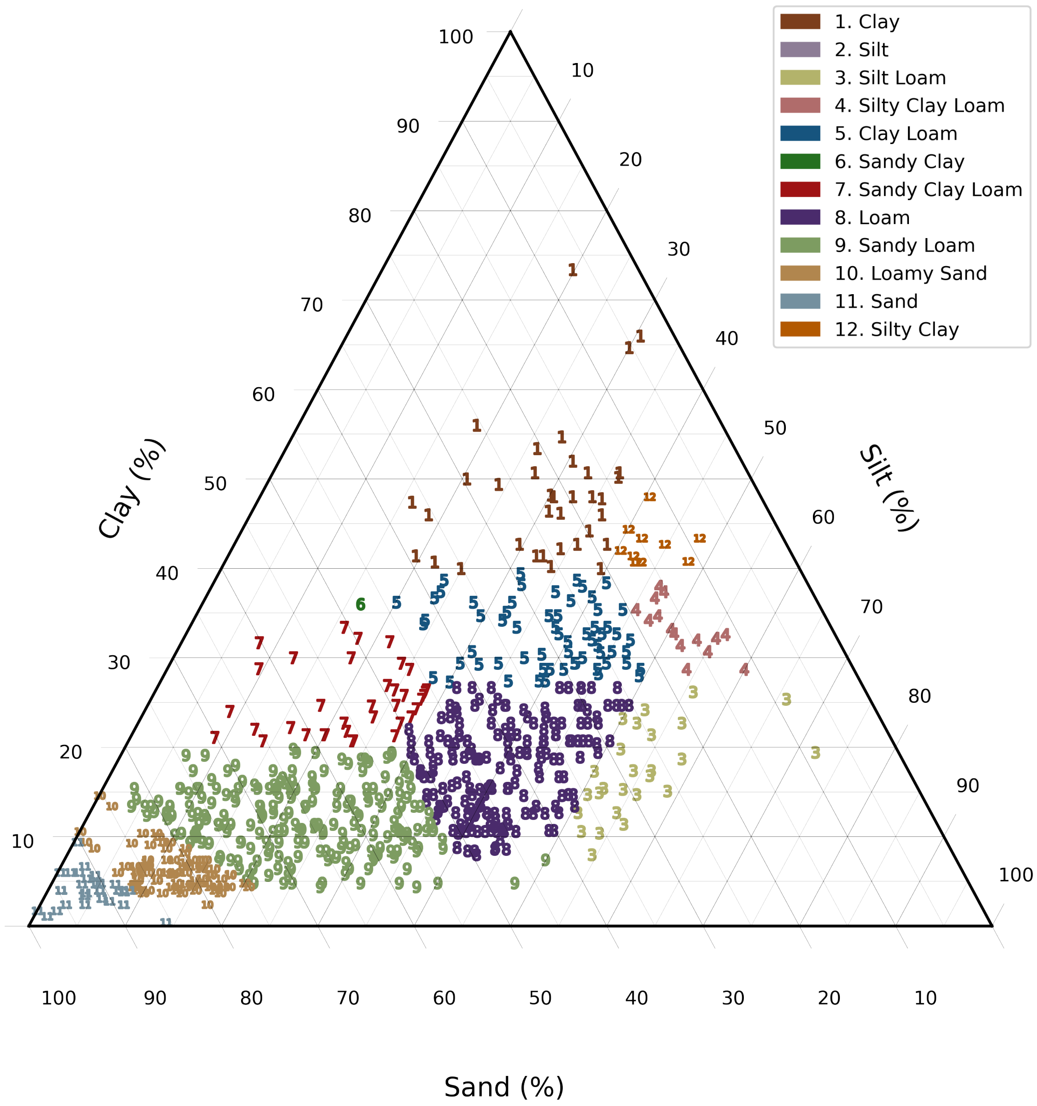


Figure 3.5: AL2019 Dataset Soil Texture Distribution

(Generated using python-ternary library)

3.5 AL2019 Dataset Compilation

In order to serve as a proper dataset well-suited for algorithmic analysis, the disparate images and metadata spread across multiple spreadsheets first needed to be assembled into a comprehensive whole, known as the *AL2019* dataset: a verified set of soil-sample image (SSI) images along with a single index containing soil texture information and all other available metadata about each one of the images.

The dataset compilation component consisted of two Python programs, *AL2019Mirror* and *AL2019Packager*, which together backed up, updated and integrated images and reference soil texture measurements into the *AL2019* dataset on a daily basis as batches were uploaded from the partner laboratory to the PASS Lab FTP server.

3.5.1 AL2019Mirror

The *AL2019Mirror* program was designed to perform a nightly backup of any new files detected on the FTP server to a Simple Storage Service (S3) server hosted by Wasabi in the cloud. This procedure served not only to safeguard the original images and metadata from loss but also to increase subsequent processing performance owing to the increased bandwidth of the S3 server as compared to the FTP server.

The *AL2019Mirror* program was scheduled for execution late every night on a GitLab cloud server. While performing the backup, it also annotated the image files stored on the S3 server with metadata about file modification times and image dimensions for inclusion in the *AL2019* dataset by the *AL2019Packager* program.

3.5.2 AL2019Packager

The *AL2019Packager* program also ran nightly, shortly after *AL2019Mirror*. Taking the files mirrored to the S3 server as inputs, it would first detect whether any new image file or Excel spreadsheet uploads had been provided. If new information was detected, the script then downloaded all of the Excel spreadsheets containing reference measurements for the soil samples from the S3 server, loaded them into memory, and merged them into a single Pandas table of soil metadata, using the unique soil sample identification numbers (IDs) as keys for the entries. Next, the soil sample entries in the table were cross-referenced with their associated image files. This was facilitated by the jFocus microscope control software, designed to include the unique IDs of soil samples in their respective image filenames upon capture (see section 3.3.4).

Having identified the metadata (reference measurements) for each image file, *AL2019Packager* then re-indexed the Pandas table by image filename, converted it to a

Python dictionary, serialized it in YAML format, and saved the result as the *AL2019* index file: a singular repository of all of the metadata in respect of the soil-sample images (SSIs), including the soil texture reference values measured at the partner laboratory using the hydrometer method. A sample entry from the index is included in Appendix D for reference.

Together, the *AL2019* index YAML file and the lossless, full-resolution (2560×1920) PNG soil-sample images constituted the complete *AL2019* dataset.

3.6 Dataset Preprocessing

3.6.1 Image Dataset Augmentation

Image transformations may be used to *augment* an image dataset by introducing, alongside its original images, variations of those images which, while differing visually from the originals, remain true to their purported content. In so doing, irrelevant features of images to which machine learning (ML) models such as CNNs might otherwise attach undue importance may be effectively averaged out and thus correctly ignored.

The range of image transformations suitable to be applied as augmentations depends on the computer vision problem being tackled. Consider, for example, a CNN being trained to distinguish images of cats from those of dogs. Human observers of such images would begin by focusing their attention on (1) the actual cats and dogs shown in the captured scenes of those images, and (2) the particular visual features of those cat and dog regions attributable to the morphology of the animals. A CNN model, however, has neither an innate ability to distinguish animal from non-animal regions of images, nor the capacity to recognize which features of those regions are morphological in nature: it sees only pixels. As such, absent an augmentation strategy designed to direct the attention of the CNN to relevant regions and features, the CNN model may latch on to irrelevant features of the cat and dog images when classifying them, resulting in poor inference performance. For example, if, for whatever reason, all (or a significant majority) of the cat images were set against a blue background whilst the dog images were set against a red background, a quite effective strategy for classification of the images would be to entirely ignore the cat or dog depicted therein and to focus instead on the color of the background – the result being, of course, that a CNN trained on such an unaugmented dataset may struggle to accurately classify images of cats unexpectedly set against red backgrounds or dogs set against blue backgrounds.

To combat such pitfalls, a number of different strategies could be considered. Ideally, care would have been taken when first capturing the images, such that factors irrelevant to the task would be steady in all cases (*e.g.* setting all cats and dogs against a same-colored background). Alternatively, however, any irrelevant features could instead be varied with a sufficient degree of randomness so as to be ignored or at least heavily discounted by the trained CNN (*e.g.* by randomly varying the background color).

It is the latter approach that an augmentation strategy via image transformations attempts to mimic *ex post facto*. Where care either was not or *could not* be taken to capture images devoid of irrelevant correlations between features and target output values, applying image transformations to randomly alter those features prior to feeding them to a ML model-in-training has the effect of systematically attenuating those correlations and diminishing their attendant predictive value. Image features contain information only to the extent that they consistently correlate with target output values; insofar as those features are uncorrelated to target output values, they will come to be ignored by ML models. Augmentation may thus be understood as a manner of guiding the learning process by suppressing the salience of irrelevant features present in a dataset by artificially varying those features at random during training.

In general, any and all irrelevant correlations between images and their associated target values which are present in the training set and which can not or should not be expected to be present in the test set are ripe for neutralization via augmentation. However, one must also not go too far when devising an augmentation strategy: care must be taken to avoid altering the images in such a way that their contents are no longer consistent with their associated target output values. For example, an image transformation consisting of artificially overlaying cat eyes on dog images is not neutral with respect to the correct target class, because such a transformation diminishes the *dogness* and increases the *catness* of the pictured animal.

3.6.2 Deterministic vs. Stochastic Image Transformations

Dataset augmentation for the DeepSoil framework was performed using the Albumentations image transformation library. Some of the transformations included in the library generate output images from input images in a deterministic fashion, such that any two executions of the transformation on the same input image would necessarily result in the same output image. For example, the `SmallestMaxSize`$(size)$ transformation is deterministic: it takes an input image with height-width-channel (HWC) dimensions $h_1 \times w_1 \times c_1$ and generates an output image with the same proportions as the input image, but resized such that the greater of its height h_2 and width w_2 is equal to $size$. As long as the input image does not change, the same output image will be generated every time. For such transformations, given an invariant input image, the output image can be computed just once and the result saved in a *cache*, such that any subsequent request to transform the same input image using the same transformation can be satisfied simply by returning the cached image, saving the processing time that would have been required to perform the transformation anew.

Other transformations, however, generate output images stochastically, *i.e.* the output image depends not only on the input image but also on the state of the program's pseudo-random number generator [82, ch. 4]. For example, the `RandomCrop`(h, w) transformation generates an output image by cropping a randomly-selected region of dimensions $h \times w$ from an input image. Subsequent requests to transform the same input image may result in different output images, such that caching the output image of the first request and returning it in response to subsequent requests would yield different results than actually performing renewed executions of the transformation.

3.6.3 Transformation Pipelines

A series of transformations $T_1, T_2, \ldots, T_n$ to be performed on an input image may be defined as a transformation *pipeline*. The first transformation T_1 takes the original image as its input, and each subsequent T_2 takes the output of the previous transformation (or pipeline *stage*) until the final transformation T_n generates the pipeline's final output image.

If all of the transformations $T_1 \ldots T_n$ are deterministic, then the entire transformation pipeline is also deterministic; that is, the pipeline's final output image will be the same for any given pipeline input image, irrespective of the random number generator state. In such case, the output image of the complete transformation pipeline can be computed in advance and cached. On the other hand, if any transformation is non-deterministic, then only the transformation steps preceding the first of the non-deterministic transformations could be cached.

3.6.4 Augmentation of AL2019 Images

The AL2019 image transformation pipeline consisted of transforming the input images by resizing them, then augmenting them via random cropping, rotation, and flipping.

Random cropping refers to an image transformation which outputs an image of smaller dimensions than its input image by randomly selecting a region of the input image. This augmentation increases the variability of the data presented to the model in training relative to using a fixed region of the image.

After cropping, images were further augmented by randomly rotating the images by either 90, 180, or 270 degrees, or not at all, at random, with equal probability, and then flipping them horizontally with a probability of 50%. The result was that for every cropped region of every image, 32 possible representations could be generated and presented to models during training and testing.

3.6.5 Random Splitting into Training and Test Sets

The final responsibility of the dataset preprocessing component was to divide the augmented images and their associated reference soil texture values into (1) a training set to be passed to the model training component and (2) a test set to be passed to the model testing component to evaluate the performance of the trained model. Splits were performed by random sampling after seeding the random number with a seed value. By explicitly setting the seed value, the random splits could be reproduced when testing different models, such that all of the models could use the same training and test sets, enabling fair comparison of their performance. A `splits` parameter defined in the presets file (Appendix H) determined the number of sequential train-test runs to execute for a given model, allowing the same model to be trained and tested using multiple combinations of training and test data. The `traintestratio` parameter, also configurable via the presets file or the command-line, was set to 0.8 for all of the tests described in this thesis, meaning that 80% (636) of the 795 images of the AL2019 dataset were used to train the CNN model, and 20% (159) were used for testing, the particular allocation of images to either the training set or the test set being dependent on the random seed set for that split.

3.7 Model Training

3.7.1 GPU Computing

Because convolutions are so common in computer graphics algorithms, dedicated hardware specialized in accelerating these operations has been developed in the form of graphics processing units (GPUs). GPUs are typically not clocked as fast as their central processing unit (CPU) counterparts, so they underperform when processing serial logic with a high degree of branching. However, GPUs often have hundreds of cores as compared with CPUs, which typically have no more than eight. The result is that GPUs excel at performing large numbers of simple operations, such as convolutions, in parallel.

In many cases, the parallel execution of operations can yield dramatic improvements in algorithmic performance. For example, whereas the edge-detection algorithm presented in section 2.3.1 computed the Sobel filter response for each region in series, a more efficient algorithm would compute the filter response for multiple or even all of the regions at the same time. A typical CPU could perform a handful of such operations in parallel, but a GPU could perform hundreds, thanks to its greater number of cores.

Given the heavy reliance of CNNs on parallel execution of convolution operations, it is often an order of magnitude faster to train CNNs using GPUs than using CPUs. For this reason, model training and testing for the DeepSoil project was primarily carried out on a Linux-based computer with a dedicated GPU (Appendix E.1.2), while simpler tests were sometimes run on the CPU of the development computer (Appendix E.1.3) for the sake of convenience.

3.7.2 PyTorch

The ability to selectively run tests on CPU or GPU was greatly facilitated by the selection of the PyTorch framework as the backbone of the dataset augmentation, model training, and model testing components of this project. PyTorch is a powerful numerical computing library for Python designed for working with multidimensional tensor data such as batches of images, and it enables virtually identical Python code to be executed interchangeably on CPU or GPU. Prototyping of simple CNNs may therefore take place on a CPU-only machine, reserving GPU resources for the training and testing of more complex CNN models with a larger number of parameters. Compared with CPU-only libraries like NumPy, performance gains can be significant [83].

The PyTorch library includes implementations of many CNN models in its `torch.models` submodule. These are typically configured to tackle classification problems, where data are assigned to one of several classes and the output values are probabilities corresponding to each class. However, soil texture prediction is not a classification problem with discrete-valued prediction outputs, but a regression problem with continuous-valued outputs, namely the predicted sand, silt, and clay proportions. Nevertheless, the same networks designed to perform classification of samples can be used for regression, the key difference lying not in how the values of the outputs are computed, but in how they are interpreted: rather than treating the models' output values as class probability predictions, the output values may be interpreted as direct estimates of the soil separate proportions.

3.7.3 Regression Loss Functions

When training a CNN, the interpretation given to its outputs is reflected in the *loss function* selected to calculate the prediction error over a set of training samples – i.e. images and reference soil texture measurements.

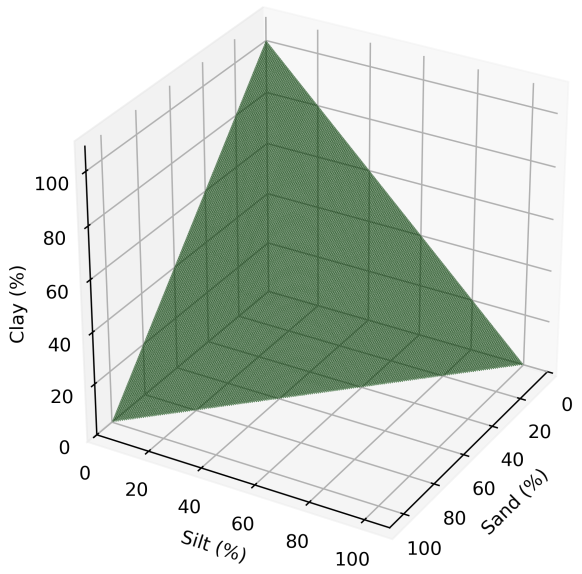


Figure 3.6: Soil Texture Space in Three Dimensions

Distance Equations

Because soil texture values have multiple dimensions, the error of a predicted soil texture $(sand_p\ silt_p\ clay_p)$ with respect to a target soil texture $(sand_t\ silt_t\ clay_t)$ is a measure of the distance between those values in a three-dimensional space (Figure 3.6). Standard distance measures amount to using a different value of p to calculate the p-norm of the distances x_i in each of n dimensions according to Equation 3.6 [84]:

$$||x||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \tag{3.6}$$

Manhattan Distance When $p = 1$, the distance is called the ℓ^1 norm, and it is computed by summing the absolute differences between the pairs of values in each dimension [84]. Owing to its grid-like appearance in space, the ℓ^1 norm is also known as Manhattan distance. The three-way soil texture ℓ^1 error between a predicted value $(sand_p\ silt_p\ clay_p)$ and a target value $(sand_t\ silt_t\ clay_t)$ is thus given by Equation 3.7:

$$||error||_1 = |sand_p - sand_t| + |silt_p - silt_t| + |clay_p - clay_t| \tag{3.7}$$

Euclidean Distance When $p = 2$, the distance is known as the ℓ^2 norm or Euclidean distance, and it is computed using Pythagoras' familiar theorem in n dimensions [84]. For three-way soil texture, the ℓ^2 norm is given by Equation 3.8:

$$||error||_2 = \sqrt{(sand_p - sand_t)^2 + (silt_p - silt_t)^2 + (clay_p - clay_t)^2} \tag{3.8}$$

Loss Functions

A *loss function* measures the aggregate error of a set of predictions by combining the errors e_i calculated as the pairwise ℓ^1 or ℓ^2 distances between predictions and target values. The loss functions most commonly used to train regression models are mean absolute error (MAE) and mean-squared error (MSE).

Mean Absolute Error The MAE is obtained by taking the mean of the absolute values of the error quantities according to Equation 3.9 [85]:

$$MAE = \frac{1}{n}\sum_{i}^{n}|e_i| \tag{3.9}$$

Mean Squared Error The MSE is the mean of the sum of squares of the error quantities according to Equation 3.10 [85]:

$$MSE = \frac{1}{n}\sum_{i}^{n}e_i^2 \tag{3.10}$$

Selected Loss Function

The DeepSoil models were trained using a loss function which consisted of taking the MAE of the ℓ^2 norm of the predicted soil texture tuples with respect to their target values. The ℓ^2 norm equates to the straight-line distance between predicted and target soil textures in the ternary soil texture plane (Figure 3.6). Selection of the MAE to aggregate the prediction errors was motivated by the desire to emphasize mean prediction accuracy. A MSE aggregation approach would have penalized large errors more strongly, encouraging more conservative predictions. Experimentation with MSE or root-mean-squared error (RMSE) may be pursued in future work.

3.7.4 Types of Models Trained and Tested

LeNet

The first type of model that was trained and tested on the AL2019 dataset was the LeNet convolutional neural network (CNN). Its small number of parameters and consequently negligible memory footprint enabled rapid training, making it ideal for early testing of the infrastructure of the DeepSoil framework. Ultimately, however, its limited expressive power led to prediction performance lagging that of deeper and more modern CNN architectures.

FFT-Based Spectral Models

Along with CNN models, several attempts were made to develop simpler custom models that might be trained more efficiently. The most successful among these was predicated on a simple approach inspired by the wavelet-based technique of Sudarsan [22]: a first stage would perform a one-dimensional Fast Fourier Transform (FFT) on each of the rows of the image to extract their frequency content; a second stage would average the frequency response across all the rows, yielding a single intermediate value for each frequency; and finally, a conventional linear regression output stage would learn to map those frequency responses to target soil texture values. Unlike the other models tested for this research, these FFT-based models were not technically CNNs, because they extracted image features using conventional, deterministic FFTs layers rather than adaptive convolutional layers. Only the output stages contained parameters which could be trained to correlate frequency responses with soil texture values.

Several variations of this approach were tried, including (1) converting the image to grayscale prior to the FFT stage, (2) performing the FFT on the rows of each color channel separately and averaging the frequency responses across channels as well as rows, and (3) employing intermediate non-linear activation elements (*i.e.* ReLUs [24]) in the final regression stage. Owing to their relative simplicity, these FFT-based networks could be trained quickly, and many did generalize sufficiently to predict soil PSDs with some accuracy, though none could achieve inference performance on par with the ResNet or MobileNetV2 models.

ResNet

The most famous ResNet CNN models are the deepest ones with a large number of layers, such as the ResNet50 and ResNet101 networks with 50 layers and 101 layers, respectively. But large networks such as these take a very long time to train and optimize unless you have access to significant computing resources. As a result, for the DeepSoil project, only the ResNet18 and ResNet50 variants were tested, and the latter only sparingly because even it was resource intensive, requiring large amounts of memory in particular. The computer used for this thesis research was limited to 4GB of GPU memory, so image batch sizes needed to be reduced when training ResNet50 models, leading to substantially slower training. Training times were probably insufficient in the ResNet50 experiments that were run, because the few ResNet50 models that were trained actually displayed inferior soil texture prediction performance relative to their smaller and more rapidly trained ResNet18 counterparts.

The long training time of ResNet50 and larger networks was exacerbated by the need to experiment with hyper-parameter values to obtain the best results. If ideal hyper-parameter values were known at the outset, the opportunity cost of delaying experiments in respect of other networks while waiting for the training of large-but-promising models to complete could have been justified. However, this was not the case – experience indicated that setting the learning rate either too high or too low, for example, could result in very slow convergence and ultimately poor prediction performance. In order to experimentally determine effective hyper-parameter values and other variables such as cropped input image dimensions, a quicker training turnover was demanded. ResNet-related research efforts for the DeepSoil project were therefore directed primarily at the smaller ResNet18 models.

MobileNetV2

The need to take resource constraints into account prompted a search for CNN architectures well-suited to less-than-state-of-the-art GPUs like that available for this project (Appendix E.1.2). Because the MobileNetV2 was developed specifically for resource constrained environments (as discussed previously in section 2.3.5), it was possible to train, test, and tune hyper-parameters for several MobileNetV2 models. These tweaks ultimately yielded the DeepSoil models with the highest soil texture prediction accuracy. A statistical analysis of their performance is presented in Chapter 4.

3.7.5 Training Algorithm

Initialization

All of the parameters governing each training run were set in a global configuration file from which sets of parameters could be selected as presets (Appendix H). Initialization began by loading the set of parameters from the presets file. The `netclass` parameter defined the type of PyTorch model to be instantiated, *e.g.* LeNet, ResNet18, MobileNetV2, or one of many custom architectures such as those based on FFTs. The parameters of the CNN model were initialized at random, and then training proceeded in a loop of steps:

1. Load a batch of size `batchsize` of augmented images and reference soil texture values from the training set.
2. Compute the output values (predicted texture) using the model.
3. Compute the loss from the output values and reference values.
4. Backpropagate the loss from the outputs to the inputs of the model.
5. Update the parameter values using the SGD optimizer.

Parameter Optimization

The loss backpropagation (4) and parameter update (5) steps were performed with the help of one of the most powerful features of PyTorch, *autograd*, which enables automatic computation of the gradients through a network [86].

The SGD optimizer's learning rate and momentum hyper-parameters were configured via the `learningrate` and `momentum` parameters of the presets file (Appendix H). Trials using two alternatives to the SGD optimizer, Adam [57] and AdamW [58], were abandoned after early experiments showed no improvement over SGD.

Termination

In some machine learning contexts, inputs to the model-in-training are the same in every epoch of training, such that the only variables are the parameters of the model itself. In such cases, after several iterations of optimization at a low learning rate, the outputs and loss will tend to stabilize. Convergence is established when the loss ceases to decline, and training can be programmatically terminated.

However, this often was not so in the DeepSoil framework. To avoid overfitting, image inputs to the CNN models were nondeterministically augmented via random cropping and rotation, and therefore the outputs computed by the model were also subject to a degree of non-determinism. This meant that while it tended to decline over time, the regression loss would never fully settle into a local minimum. Even when the learning rate had been scheduled to decline to a very low value, the output values continued to bounce around, sometimes dipping to new lows, other times jumping a little higher. This complicated the task of determining when to terminate training.

Three termination strategies were employed: first, setting a maximum number of training *epochs*; second, setting a maximum training *time*; and third, detecting convergence by measuring the amount of variability in the loss. The last strategy ultimately proved unfruitful, as variations in the loss often exceeded the very gradual loss reduction that trended almost imperceptibly downward over the course of several epochs. Like a surfer riding the waves as the tide goes out, the peaks and troughs rendered the general decline imperceptible. It may have been possible to overcome this difficulty by computing moving averages of the loss, but the potential benefit of implementing this was outweighed by the required investment in time and energy. Instead, it was decided to fall back on the simpler strategies of terminating training either after a fixed number of epochs or a fixed amount of time.[3]

3.8 Model Testing

The model testing component loaded the CNN model trained by the model training component and used it to predict soil texture values for each of the images of the test set generated by the dataset preprocessing component. In order to provide the model with input tensors consistent with those of the training data, the same types of image transformations that were performed on the training set images were again performed on the test set images. For example, where a model had been trained on images which had been resized by a certain factor, then cropped and rotated at random, those same operations were performed on the images of the test set before feeding them to the model as inputs.

3.8.1 20-Sample Average

However, one important difference existed between the training and test phases. Whereas, in training, the CNN model would predict output values on the basis of a single sampled (cropped) region from each input image, in testing the model was allowed to see 20 sampled regions from the image and make its final prediction using the mean of the output predictions over the 20 inputs. It was hoped and expected that this strategy would increase prediction accuracy by averaging out random noise in the outputs generated, on the assumption that at least some of the error in any one prediction would be independent from that in one or more of the others, such that the mean prediction might tend to average out the noise, thus highlighting the prediction 'signal'. As will be discussed in section 4.3.2, there are indications that this expectation was not without merit.

[3]For convenience, an additional user-initiated termination feature was also added, whereby training could be prematurely terminated by pressing the `Ctrl-C` keys to send the model training program a `SIGINT` signal which would be received as a `KeyboardInterrupt` exception in Python.

3.9 Statistical Evaluation

Beyond the development of the image acquisition, dataset compilation, preprocessing, and model training and testing components described above, a final aspiration of the DeepSoil project was to identify which CNNs model architectures demonstrated the most promise at effective prediction of soil texture from digital microscope SSIs. This motivated the development of one last component: a statistical evaluation system which could be used to compare the training and testing performance of hundreds of combinations of models and hyper-parameter values.

3.9.1 Implementation

The statistical evaluation component was designed as a combination of a `stats` Python module and a Jupyter notebook – a graphical interactive Python shell allowing for visualization of the information gathered. The model training and testing components were adapted to write statistically-relevant information to disk in YAML files for each train-test run. A simple Python script was then written to merge the data generated by all of the separate train-test runs into a single database and save it as a YAML file. The Jupyter notebook then loaded the file containing all of the gathered information regarding the train-test runs and evaluated their relative performance on the basis of statistical metrics.

3.9.2 Metrics

Three statistical metrics were computed to assess the prediction performance of the tested models: mean prediction error, median prediction error, and coefficient of determination.

Mean Prediction Error The mean distance between predicted and reference soil texture values for each of the sand, silt, and clay components, as well as the ℓ^2 norm of the three distance components.

Median Prediction Error The median distance between predicted and reference soil texture values for each of the sand, silt, and clay components, as well as the ℓ^2 norm of the three distance components.

Coefficient of Determination The coefficient of determination (R^2) of a model quantifies the degree to which it is able to account for the variance in a set of target values y_i for $i \in [1, n]$. The variance of the values y is a measure of how dispersed they are about their mean $\bar{y}$ according to Equation 3.11 [87]:

$$\sigma^2 = \frac{1}{n}\sum_{i}^{n}(y_i - \bar{y})^2 \tag{3.11}$$

The *residuals* of the model are the pairwise distances between the model's predictions $\hat{y}$ and the reference values y. The coefficient of determination measures the proportional decrease in the sum of square residuals versus the sum of square differences between the reference values and their mean according to Equation 3.12 [87]:

$$R^2 = 1 - \frac{\Sigma_i (y_i - \hat{y}_i)^2}{\Sigma_i (y_i - \bar{y})} \tag{3.12}$$

The R^2 value therefore measures the proportional decrease in the sum of square errors which result from using the model instead of the mean to predict output values. Put simply, the R^2 of a model represents its explanatory power.

Chapter 4. Results and Discussion

4.1 Model Performance Comparison

To compare model performance and identify which one was most effective, the collected statistics about the train-test runs were grouped according to the texture type (three-way or four-way) and sorted in descending order by mean ℓ^2 prediction error computed over 10 splits of the AL2019 dataset. Results were then printed out and visualized using the statistical evaluation component's interactive Jupyter notebook. A summary of the results is presented in Table 4.1.

Model	Test Error (ℓ^2)		Training Error (ℓ^2)
	Mean	Median	μ (*last* 5 *epochs*)
`al2019_threeway_mobilenetv2_300_300`	0.132	0.108	0.095
`al2019_threeway_mobilenetv2_300_280`	0.135	0.108	0.103
`al2019_threeway_mobilenetv2remainder_300_300`	0.139	0.111	0.107
`al2019_threeway_mobilenetv2_600_300`	0.140	0.115	0.142
`al2019_threeway_mobilenetv2_400_400`	0.145	0.117	0.117
`al2019_threeway_resnet18_600_300`	0.156	0.130	0.166
`al2019_threeway_resnet18_600_400`	0.157	0.127	0.161

Table 4.1: Top 7 three-way AL2019 soil texture models

The `al2019_threeway_mobilenetv2_300_300` model was the best-performing model for three-way texture prediction, exhibiting the lowest mean prediction error. Details of the training error, mean test error, and median test error for this model will be further discussed below.

4.2 Training Statistics

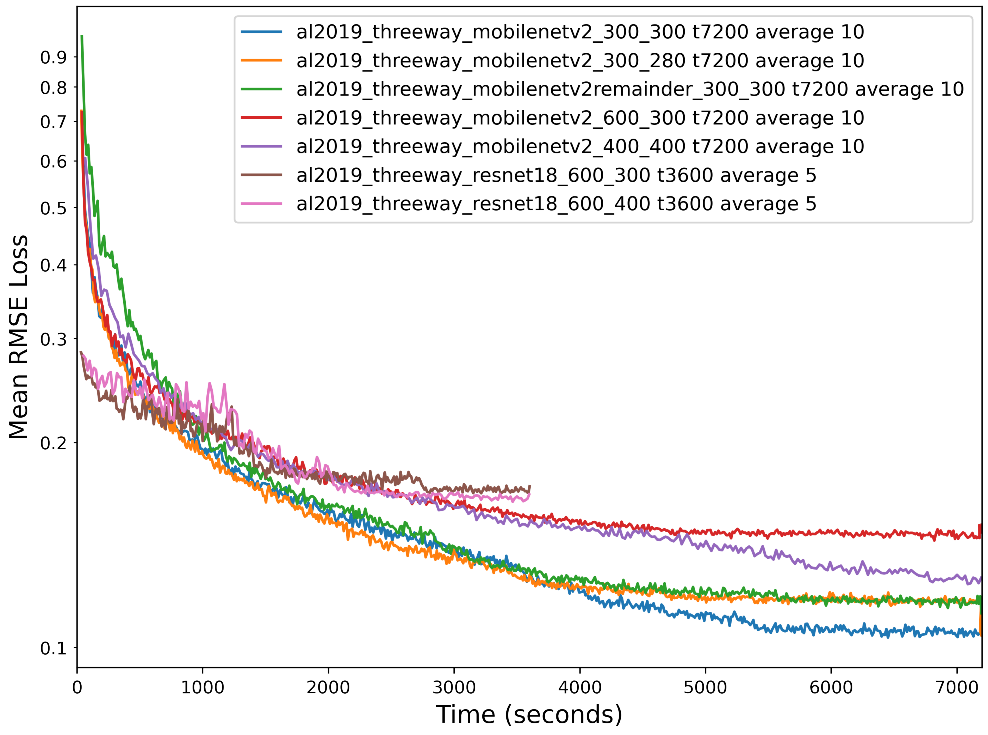


Figure 4.1: Training Loss vs. Time

Figure 4.1 summarizes the learning process undergone by each of the top 7 three-way texture models. The value of the loss function, averaged over each epoch, is plotted versus time trained using the GPU of the model training and testing computer (Appendix E.1.2). The five MobileNetV2 models were trained for 7200 seconds (2 hours) for each of 10 random splits of the AL2019 dataset, while the two ResNet18 models were trained for only 3600 seconds (1 hour) and 5 splits each. The reason for the discrepancy in training time was that efforts were concentrated on the MobileNetV2 models, which – as can be seen in the graph – exhibited better learning performance within the first 3600 seconds of training. Training time appropriate for each model was assessed based on empirical observation of the flattening of the training curve in plots such as that shown. Notably, the curve of the `al2019_threeway_mobilenetv2_400_400` had not completely flattened at the 7200-second mark, suggesting that additional training may have further reduced training loss and potential test prediction error. Because this model lagged others at the 7200-second mark, resources were not dedicated to this inquiry for this thesis, though it may be worth pursuing in future work.

4.3 Test Statistics

4.3.1 Absolute Prediction Error

Table 4.2 shows absolute prediction error statistics for the best-performing of the DeepSoil models, `al2019_threeway_mobilenetv2_300_300`, over 10 splits of the AL2019 dataset, where each split was performed at random after seeding the random number generator with the seed value indicated in the first column. The mean (μ) and standard deviation (σ) of these errors over the 10 splits is also shown, giving a better view of the robustness of the model with respect to different training and test sets. The indicated quantities are unitless, since they represent the deviation of the predicted sand, silt, and clay proportions from those measured using the hydrometer method in absolute terms (*i.e.* in kilograms per kilogram).

Seed	Mean Absolute Error				Median Absolute Error				R^2			
	ℓ^2	$sand_3$	$silt_3$	$clay_3$	ℓ^2	$sand_3$	$silt_3$	$clay_3$	ℓ^2	$sand_3$	$silt_3$	$clay_3$
0	0.124	0.089	0.057	0.048	0.095	0.066	0.040	0.036	0.653	0.652	0.730	0.448
1	0.139	0.098	0.065	0.055	0.110	0.067	0.049	0.037	0.597	0.610	0.655	0.441
2	0.130	0.087	0.057	0.059	0.105	0.067	0.039	0.046	0.681	0.725	0.686	0.542
3	0.133	0.093	0.068	0.046	0.118	0.081	0.059	0.038	0.672	0.689	0.647	0.652
4	0.130	0.092	0.063	0.049	0.097	0.064	0.048	0.033	0.672	0.685	0.677	0.616
5	0.133	0.092	0.062	0.054	0.105	0.070	0.047	0.034	0.649	0.687	0.620	0.567
6	0.137	0.096	0.069	0.050	0.124	0.083	0.056	0.034	0.653	0.672	0.635	0.609
7	0.132	0.091	0.058	0.058	0.107	0.069	0.046	0.041	0.666	0.685	0.723	0.517
8	0.127	0.089	0.060	0.050	0.107	0.068	0.045	0.039	0.682	0.704	0.691	0.589
9	0.139	0.094	0.065	0.058	0.116	0.074	0.050	0.038	0.676	0.712	0.687	0.548
μ	0.132	0.092	0.062	0.053	0.108	0.071	0.048	0.038	0.660	0.682	0.675	0.553
σ	0.005	0.003	0.004	0.004	0.009	0.006	0.006	0.004	0.024	0.031	0.034	0.066

Table 4.2: Absolute error of `al2019_threeway_mobilenetv2_300_300` predictions for 10 splits of the AL2019 dataset

Mean Prediction Error As shown in Table 4.2, the mean ℓ^2 norm of prediction error for the best DeepSoil model was 0.132. The mean of the prediction errors for individual texture components were 0.092 for sand, 0.062 for silt, and 0.053 for clay.

Median Prediction Error In the median case, prediction errors were lower, with a mean ℓ^2 norm error of 0.108 and individual errors of 0.071 for sand, 0.048 for silt, and 0.038 for clay. The low median error relative to median error may be explained in part by the selection of the MAE loss function, which, relative to the MSE loss function, de-emphasizes outliers. Training the model using MSE instead of MAE loss may have resulted in reduced mean prediction error at the expense of greater median prediction error.

Coefficient of Determination The final columns of Table 4.2 indicate the coefficient of determination (R^2) statistics for the `al2019_threeway_mobilenetv2_300_300` model for each one of the 10 splits of the AL2019 dataset. On average, R^2 was 0.660 for the ℓ^2 norm, 0.682 for sand, 0.675 for silt, and 0.553 for clay.

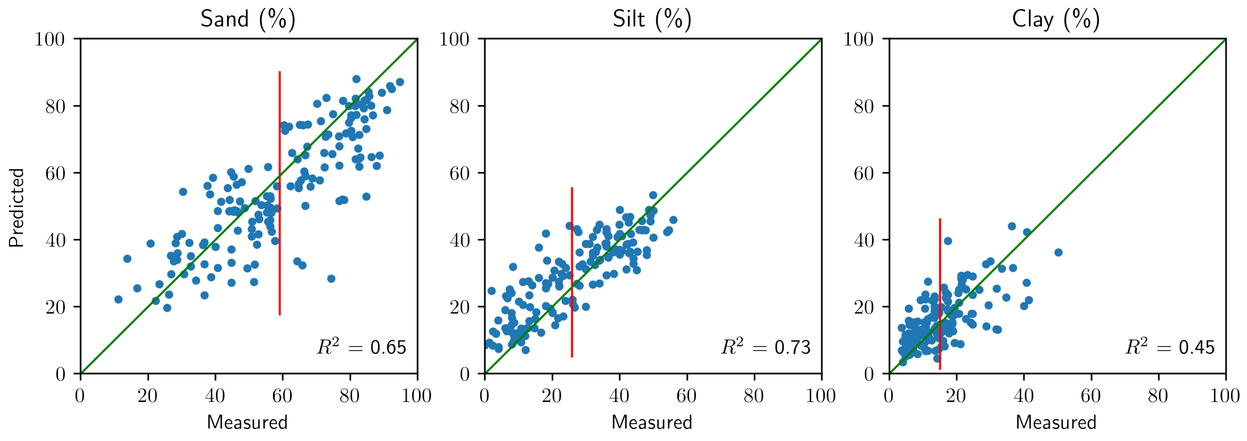


Figure 4.2: Predicted vs. Measured Values ($Seed = 0$)

The significance of the coefficient of determination (R^2) metric may be visually perceived in Figure 4.2, which shows the predicted soil separate proportions plotted with respect to their corresponding reference values for the first split of the AL2019 dataset (*i.e.* with $Seed = 0$). Recall that, according to Equation 3.12, the coefficient of determination is equal to one minus the ratio between the sum of squared *residuals* (model prediction errors) and the sum of squared deviations of the measured values from their mean. In the subplots of Figure 4.2, the diagonal green lines represent hypothetical perfect predictions, and the *residuals* are equal to the vertical distances separating each one of the blue sample points from those diagonals. The mean of the measured values is indicated as a vertical line in each subplot, and the *deviations* are the horizontal distances separating the sample points from that line. In other words, the coefficient of determination expresses the degree to which sample points are nearer (in the vertical direction) to the green diagonal than they are (in the horizontal direction) to their mean (red line).

More precisely, the coefficient of determination quantitatively summarizes the degree to which model predictions explain the variance of the measured values. Where sample points are generally nearer to the diagonal of perfect prediction than they are to the mean measured value, the effective variance which remains in light of the model's predictions is reduced, and the coefficient of determination is accordingly high (*e.g* the silt subplot with $R^2 = 0.73$). Conversely, where model predictions offer a lesser decline in variance, the coefficient of determination is lower (*e.g.* the clay subplot with $R^2 = 0.45$).

4.3.2 Generalization

The correspondence between the training loss achieved by fitting the model to the training data and the subsequent test loss when making predictions using the test data indicates how well the training strategy was able to *generalize*. The last column of Table 4.1 shows the mean training loss, taken over the last 5 epochs of training to reduce noise.

As one would expect, the training error for most of the models is lower than the test error because the model had the benefit of knowing the training reference values and adapting its parameters to make accurate predictions, whereas it could not make any adjustments on the basis of the test data.

However, for some of the models – notably, both ResNet18 models – the mean test error was actually lower than the training error. This might be explained by the use of multi-sample averages when making predictions during testing versus single-sample predictions in training. That is, whereas predictions were based on a single cropped region sampled from each image during training, each final prediction in testing was formed by taking the mean of the model outputs over 20 cropped regions of the input image. One possible explanation for the decrease in test error relative to training error may be that the 20-region mean prediction strategy succeeded in detecting a greater number of relevant features than the individual single-region predictions used to train the model. In future research, a statistical evaluation could be performed to evaluate the degree to which the accuracy of the 20-sample mean prediction surpassed that of each one of the 20 single-sample predictions on which it was based.

4.3.3 Relative Prediction Performance

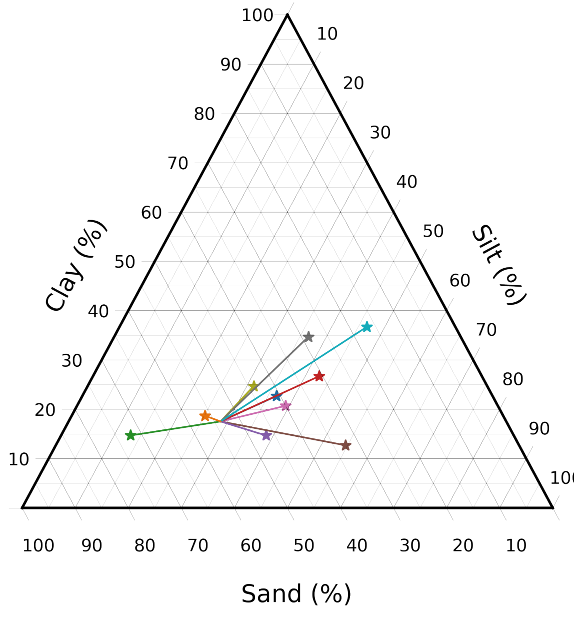
The prediction error metrics above describe the accuracy of the DeepSoil models in absolute terms, but the real value of these models ought to be contemplated on a relative basis. Consider that, absent any soil-sample image data whatsoever, a much simpler *baseline* model could make soil texture predictions for the samples of the test set using only the soil texture values of the training set as training data. For example, this baseline model could simply compute the mean texture value over the training set and use that mean value as its prediction for all of the soil samples of the test set. How would the absolute prediction performance of such a model compare to that of the DeepSoil models?

Seed	Prediction			Mean Absolute Error				Median Absolute Error			
	$sand_3$	$silt_3$	$clay_3$	ℓ^2	$sand_3$	$silt_3$	$clay_3$	ℓ^2	$sand_3$	$silt_3$	$clay_3$
0	0.538	0.286	0.175	0.242	0.176	0.131	0.074	0.242	0.171	0.134	0.068
1	0.546	0.281	0.173	0.255	0.188	0.129	0.086	0.264	0.190	0.128	0.067
2	0.557	0.277	0.166	0.256	0.190	0.115	0.099	0.245	0.190	0.115	0.089
3	0.549	0.280	0.172	0.235	0.170	0.121	0.081	0.217	0.153	0.116	0.068
4	0.548	0.283	0.170	0.259	0.194	0.125	0.090	0.250	0.185	0.123	0.079
5	0.552	0.280	0.168	0.244	0.179	0.114	0.091	0.236	0.175	0.106	0.068
6	0.548	0.280	0.172	0.247	0.183	0.122	0.084	0.231	0.162	0.116	0.078
7	0.552	0.280	0.168	0.251	0.185	0.120	0.089	0.265	0.189	0.120	0.075
8	0.543	0.286	0.171	0.252	0.188	0.123	0.088	0.252	0.186	0.125	0.076
9	0.556	0.277	0.166	0.273	0.201	0.132	0.094	0.271	0.198	0.134	0.072
μ	0.549	0.281	0.170	0.252	0.185	0.123	0.088	0.247	0.180	0.122	0.074
σ	0.005	0.003	0.003	0.010	0.009	0.006	0.007	0.016	0.013	0.008	0.007

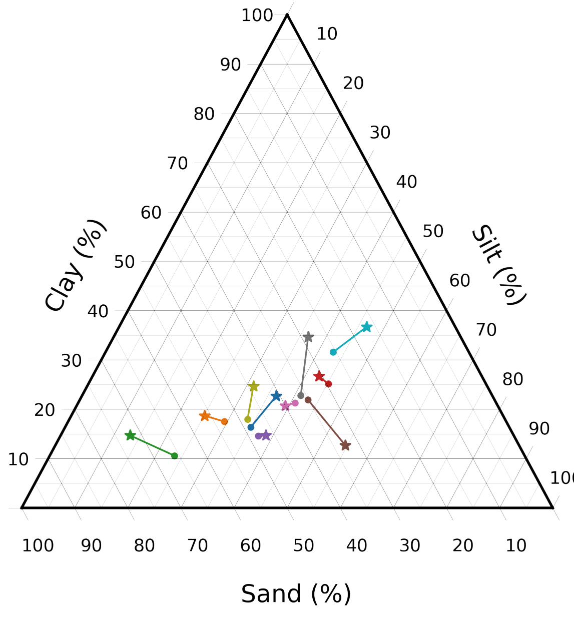
Table 4.3: Baseline error for 10 splits of AL2019 dataset

In Table 4.3, the prediction value which results from taking the mean of the training set is given in a first set of columns, followed by the mean and median absolute prediction errors that result from using this prediction over the test set. The experiment was repeated over the same 10 splits of the AL2019 dataset as for the results of the best known DeepSoil model, `al2019_threeway_mobilenetv2_300_300`. The mean ℓ^2 error was 0.252 versus the 0.132 mean ℓ^2 error of the DeepSoil model – in other words, the DeepSoil model's soil texture predictions were nearly twice as accurate. The outperformance of the DeepSoil `al2019_threeway_mobilenetv2_300_300` model was even greater in the *median* case, with median prediction error of only 0.108 versus the baseline model's 0.247.

The relative performance of the model and baseline predictions is represented visually in Figure 4.3, with predicted values indicated by circular markers connected by lines to their respective reference values, denoted by star-shaped markers. As can be seen in subfigure 4.3a, the baseline model uses the training set mean texture value as its prediction for all of the test set samples. In subfigure 4.3b, the model has learned to make better predictions, resulting in shorter lines from predicted to reference values.



(a) Baseline predictions



(b) Model predictions

Figure 4.3: `al2019_threeway_mobilenetv2_300_300` model predictions versus baseline predictions for 10 randomly selected samples

4.4 Limitations and Assumptions

The inference performance of a machine learning model trained on a particular set of data points will degrade if the distribution of test data points departs from that on which it was trained. The DeepSoil implementation was predicated on the assumption that the soil samples of the AL2019 dataset are representative of the population of soil samples for which soil texture predictions would be made. As indicated above in Figure 3.5 and Table 3.3, the AL2019 dataset's distribution is somewhat biased towards loamy and sandy soils, given that those are the types of soils for which soil texture analysis were most often requested by customers of A&L Canada. Insofar as future soil samples under analysis are similarly distributed, it is expected that the inference performance of the DeepSoil models presented herein would continue to hold. However, if, for whatever reason, soil samples following a different distribution were analyzed, such as a distribution skewed toward higher clay or silt content, then the inference performance of DeepSoil models trained on the AL2019 dataset would likely suffer.

At least two strategies might be employed to overcome this limitation, should it be of concern. First, new soil images could be added to the dataset for samples whose reference (e.g. hydrometer) measurements are distributed more evenly across the full range of soil textures which the prediction model is expected to encounter. Second, when training the model, the contribution of soil texture data points to the loss function could be weighted according to their relative distance from the mean texture value, such that the influence over the model parameters of the points nearer to the mean value would be discounted whilst points more distant from the mean would carry more weight, thereby artificially replicating the behaviour of a more even training distribution. This strategy would be analogous to that of employing a weighted cross-entropy loss function for a classification problem [88].

However, absent a basis for expecting a departure of the distribution of soil textures in a test set from those of the training set (*i.e.* the AL2019 dataset), efforts to compensate for the idiosyncrasies of the training data should be avoided, as they would likely result in *reduced* prediction accuracy. The distribution of soil textures present in the training set is valuable information in itself, which, under normal circumstances, should be taken into account by the model. Indeed, using this information alone, baseline predictions of soil texture were made with a mean ℓ^2 error of just 0.252 over ten splits (see Table 4.3).

Chapter 5. Future Research

While the results obtained using the methods discussed above were good, further performance improvements may be achievable. In this chapter, several potential avenues for future research are presented.

5.1 Specialized CNN Architectures

Because of time and computing resource constraints, it was impossible to try all of the known CNN architectures and evaluate their relative training and testing performance. There are many more model architectures that could be used to predict soil texture from microscope images. Two stand out as particularly strong candidates: Fisher vector-based CNNs and EfficentNets.

5.1.1 FV-CNN

Conceptually, a correct model for evaluating soil texture from images should not care where the particles present in the image are located, but only that they appear *somewhere* in the image. As described in section 2.3.5, a Fisher vector CNN uses FV pooling layers to abstract out the spatial locations where features are detected, making it better suited than fully-connected CNNs for tasks such as image texture analysis, where the location of features within the image are not relevant [70]. These conditions also characterize digital microscope images of soil samples, and there is therefore reason to believe that FV-CNNs would excel at soil texture prediction.

5.1.2 EfficientNet

Like the MobileNetV2 architecture that ultimately proved most successful for this thesis research, EfficientNet CNN designs specifically target resource-constrained contexts.

Unless significantly more resources are available for future DeepSoil research, it may be worth exploring whether an EfficientNet could outperform the MobileNetV2 model at AL2019 PSD prediction. The authors of the EfficientNet paper propose a range of models with different trade-offs [69]. The standard PyTorch library does not include an EfficientNet implementation, but a third-party Python package is in active development.

5.2 Higher Resolutions

In Table 4.2, the R^2 performance in respect of clay of the best DeepSoil model, `al2019_threeway_mobilenetv2_300_300`, lagged with respect to prediction of the sand and silt components. This is unsurprising given that the images on which this model was based were resized from the original (2560×1920) resolution down to (400×300), and clay particles were theoretically no more than 3 pixels in diameter in the original image. Limited experimentation with higher resolution models was performed during this research because larger images put more pressure on memory resources. But given that clay particles are theoretically invisible in small images, there is reason to believe that further experimentation with higher resolution models and image transformations would improve clay prediction performance. Moreover, given that texture values are ternary, better prediction of the clay component necessarily implies better prediction of the non-clay component, therefore sand and silt proportions also ought to be predicted more accurately.

5.3 Multiple Magnification Levels

At the other end of the size spectrum, it may be advisable to use multiple images of soil samples, each taken at a different magnification level, in order to better capture sand particles. In this thesis research, the magnification level was set to the maximum possible value so as to render clay particles as visible as possible. In so doing, the physical dimensions of the region displayed in any one image were $1.71mm \times 1.28mm$. Because sand particles may be up to $2mm$ in diameter, some particles could theoretically exceed the dimensions of the image, rendering them altogether invisible in the worst case. In order to capture both clay particles at one end of the size spectrum and sand particles at the other, the AL2019 dataset could be augmented with further images of soil samples taken at a lesser magnification level, perhaps even using a regular digital camera rather than a microscope. The use of multiple images taken at greater and lesser magnification levels for each soil sample theoretically ought to enable greater visibility of soil particles across the spectrum from clay to sand, and perhaps even gravel.

5.4 Hyper-parameter Refinements

For this research, hyper-parameters were manually adjusted in an *ad hoc* fashion based on observed increases in the convergence rate of the training loss and the inference performance of trained networks. A more systematic approach to setting and experimenting with hyper-parameter values may yield better results.

5.4.1 One-Cycle Learning Rate Scheduling

Gugger [89], echoing Smith [90], suggests using a "one-cycle" learning rate scheduling strategy in which the learning rate is first set to a low value, then gradually incremented in each epoch until it reaches the largest (estimated) value that avoids divergence, before finally being decreased again to a low value. The logic of this approach is consistent with patterns in the training loss which were observed while training the DeepSoil models. The magnitude of the gradient tends to be high in the first few epochs of training, and the model parameters adjust quickly from the random values with which they were initialized. Applying a high learning rate in the first few epochs of training therefore risks causing parameter value updates that are too large, driving the loss function towards divergence. Conversely, a smaller initial learning rate allows the model's natural tendency to learn in the first few epochs to play out naturally. After this initial phase, the gradient of the loss function tends to decline, and a higher learning rate would encourage faster training – Bengio [55] has observed that the optimal learning rate is usually within a factor of 2 of the largest learning rate that does not cause divergence of the training loss. Finally, when training is winding down, a lower learning rate will allow the model to converge nearer to the local optimum in which it is settling.

5.4.2 Exploring the Hyper-parameter Space

There are already a vast number of variables involved when training CNNs, such as the model architecture and image augmentations, before even considering all of the hyper-parameters used in training such as learning rate, momentum, batch size and learning rate schedule. Each combination takes time to test, especially if computing resources are limited, and not all combinations can be tested because there are too many. A strategy is therefore required to efficiently explore the hyper-parameter space. One possibility is a grid search, where a set of potential values for each hyper-parameter is determined, and all of the combinations are tested, but it has been shown that setting hyper-parameter values at random is more efficient [91].

5.5 Parallel Training of Models

For reasons of simplicity and cost efficiency, this thesis project was executed on a single self-hosted desktop computer equipped with an inexpensive GPU (Appendix E.1.2). A significant limitation of this approach was that each set of hyper-parameters had to be evaluated in series, rendering the type of hyper-parameter search described above impractically time-consuming.

Fortunately, the problem of identifying effective hyper-parameter candidates is eminently parallelizable – there are no dependencies between executions of the training algorithm with respect to hyper-parameter values. This means that various combinations of hyper-parameter values could be tested in parallel on different machines, if those machines were available. The time required to execute the hyper-parameter search would then be inversely proportional to the number N of processing units (CPUs or preferably GPUs) used.

5.5.1 Self-Hosted Servers

Obvious performance gains could therefore be obtained simply by adding more processing units to a self-hosted architecture – *i.e.* by adding CUDA-compatible GPU cards to an existing computer and/or adding additional computers. These could include conventional Intel or AMD-based desktop computers with GPU cards, or low-cost computers targeted specifically to the machine learning market, such as NVIDIA's Jetson[1] or Google's Coral[2] product lines.

5.5.2 Cloud Servers

Cloud-based virtual servers offer a more flexible solution. Of particular interest are the GPU-enabled server instances now offered by many companies, such as Google Cloud,[3] Amazon EC2,[4] and newer entrants like Paperspace.[5] An important advantage of cloud solutions is their relative reliability and ease of maintenance: professionally-managed cloud servers are – or at least *ought* to be – less failure-prone than self-hosted computers, and save users from the hassles of installing or updating drivers, debugging network connectivity, or rebooting to recover from failure. Of course, service levels and pricing models vary, and a cost-benefit analysis would need to be carried out to compare cloud service providers as well as self-hosted options.

5.6 Systematic Comparison of Models and Parameters

Because the objective of the research presented herein was to assess the viability of employing deep learning to predict soil texture from digital microscope images, its scope was limited to identifying a handful of models with satisfactory performance, in terms of both predictive ability and training efficiency (given resource limitations), as opposed to the *best* possible model. As previously explained, investigation of different

[1] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/
[2] https://coral.ai/products/
[3] https://cloud.google.com/gpu/
[4] https://aws.amazon.com/ec2/instance-types/
[5] https://www.paperspace.com

models and fine-tuning of hyper-parameter values demand a significant investment in hardware resources and time, as each model needs to be trained, tested, and statistically evaluated for each tweak of each hyper-parameter value. The task of systematically and quantitatively comparing the performance of multiple models and hyper-parameter values would constitute a serious research effort worthy of a future thesis project unto itself. Were such research to be undertaken, it would be strongly advised to parallelize the model and hyper-parameter search, as described in the previous section, to save time.

5.7 Web Interface

With relatively little effort, a web-based interface could be added to the DeepSoil framework to facilitate remote submission of additional soil sample images. For example, the interface could be set up as a REST API running on a server in the cloud that would accept structured JavaScript Object Notation (JSON) requests each containing a soil-sample image captured with a digital microscope and any other relevant metadata such as the model and magnification level of the microscope.

In a first use case, the provided image would be fed as input to a trained DeepSoil CNN model(s), which would then compute a predicted soil texture value to be included in the structured JSON reply to the client request.

In another use case, the request metadata could further include a reference texture value associated with the provided image, such that the request would provide a new labelled training data point which could be used to further train and refine the predictive accuracy of the DeepSoil CNN model(s).

In concert with remote collection of soil samples and capture of digital microscope images, such a web interface would convert the DeepSoil research framework into a practical and possibly commercially-viable tool for *remote* assessment of soil texture.

Chapter 6. Conclusion

The objective of this thesis was to investigate the suitability of convolutional neural networks for soil particle size analysis. A deep-learning system comprising many components was developed to perform all of the tasks required, from controlling a digital microscope via custom software to capture high-quality images, to assembling and transforming the images and associated metadata into a comprehensive dataset, to training and testing a wide range of CNN models tuned according to a variety of hyper-parameter values, before finally performing a statistical evaluation of the results of these experiments.

The best performing of these models was able to predict sand, silt, and clay content with mean absolute errors of 9.2%, 6.2%, and 5.3%, respectively, and mean absolute error of the ℓ^2 norm of 13.2%. Median absolute prediction errors were even smaller: 7.1%, 4.8%, and 3.8% for sand, silt, and clay, and 10.8% for the ℓ^2 norm. In the agricultural context, these soil texture estimates are sufficiently accurate to usefully inform farm practices and policies.

Future research promises to further improve model accuracy and availability via the web. With these models having already been trained, the marginal cost of making new predictions is low: only a low-cost digital microscope, a 3D-printable microscope holder, a personal computer, and an Internet connection are required. As such, over the longer term, deep learning approaches such as those investigated in the DeepSoil framework promise to reduce the cost and increase the availability of soil particle size analysis, which, together with smart resource management, may ultimately lead to improved outcomes for producers and society more broadly.

References

[1] G. Rapsomanikis, “The economic lives of smallholder farmers: An analysis based on household data from nine countries,” *Food and Agriculture Organization of the United Nations, Rome*, 2015 (cit. on p. iii).

[2] R. Gebbers and V. I. Adamchuk, “Precision agriculture and food security,” *Science*, vol. 327, no. 5967, pp. 828–831, Feb. 12, 2010, ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1183899 (cit. on p. 1).

[3] F. M. Van Egmond, E. H. Loonstra, and J. Limburg, “Gamma ray sensor for topsoil mapping: The mole,” in *Proximal Soil Sensing*, Springer, 2010, pp. 323–332 (cit. on pp. 1, 7).

[4] V. I. Adamchuk, J. W. Hummel, M. T. Morgan, and S. K. Upadhyaya, “On-the-go soil sensors for precision agriculture,” *Computers and electronics in agriculture*, vol. 44, no. 1, pp. 71–91, 2004 (cit. on p. 1).

[5] R. A. Viscarra Rossel, S. R. Cattle, A. Ortega, and Y. Fouad, “In situ measurements of soil colour, mineral composition and clay content by vis–NIR spectroscopy,” *Geoderma*, vol. 150, no. 3, pp. 253–266, May 15, 2009, ISSN: 0016-7061. DOI: 10.1016/j.geoderma.2009.01.025 (cit. on pp. 1, 6).

[6] W. Ji, V. I. Adamchuk, S. Chen, A. S. M. Su, A. Ismail, Q. Gan, Z. Shi, and A. Biswas, “Simultaneous measurement of multiple soil properties through proximal sensor data fusion: A case study,” *Geoderma*, vol. 341, pp. 111–128, 2019, Publisher: Elsevier (cit. on p. 1).

[7] P. A. Sanchez, S. Ahamed, F. Carré, A. E. Hartemink, J. Hempel, J. Huising, P. Lagacherie, A. B. McBratney, N. J. McKenzie, and M. de Lourdes Mendonça-Santos, “Digital soil map of the world,” *Science*, vol. 325, no. 5941, pp. 680–681, 2009 (cit. on p. 2).

[8] D. Hillel and J. L. Hatfield, *Encyclopedia of Soils in the Environment*. Elsevier Amsterdam, 2005, vol. 3 (cit. on p. 2).

[9] J. Selker and D. Or, *Soil Hydrology and Biophysics*. Oregon State University (cit. on pp. 4–6, 78).

[10] G. W. Gee and D. Or, "Particle-size analysis," in *Methods of soil analysis: Part 4 physical methods*, vol. 5, Publisher: Wiley Online Library, 2002, pp. 255–293 (cit. on pp. 4–6).

[11] W. P. Miller and D. M. Miller, "A micro-pipette method for soil mechanical analysis," *Communications in Soil Science and Plant Analysis*, vol. 18, no. 1, pp. 1–15, 1987 (cit. on p. 6).

[12] G. J. Bouyoucos, "A recalibration of the hydrometer method for making mechanical analysis of soils 1," *Agronomy journal*, vol. 43, no. 9, pp. 434–438, 1951 (cit. on p. 6).

[13] ——, "Hydrometer method improved for making particle size analyses of soils 1," *Agronomy journal*, vol. 54, no. 5, pp. 464–465, 1962 (cit. on p. 6).

[14] H. Shimaoka, "Particle size analyzer based on laser diffraction method," pat. 6 473 178, Publisher: Google Patents, Oct. 29, 2002 (cit. on p. 6).

[15] M. Panalytical. (2020). Mastersizer laser diffraction particle size analyzer, [Online]. Available: https://www.malvernpanalytical.com/en/products/product-range/mastersizer-range (visited on 06/23/2020) (cit. on p. 6).

[16] M. Nocita, A. Stevens, B. van Wesemael, M. Aitkenhead, M. Bachmann, B. Barthès, E. Ben Dor, D. J. Brown, M. Clairotte, A. Csorba, P. Dardenne, J. A. M. Demattê, V. Genot, C. Guerrero, M. Knadel, L. Montanarella, C. Noon, L. Ramirez-Lopez, J. Robertson, H. Sakai, J. M. Soriano-Disla, K. D. Shepherd, B. Stenberg, E. K. Towett, R. Vargas, and J. Wetterlind, "Chapter four - soil spectroscopy: An alternative to wet chemistry for soil monitoring," in *Advances in Agronomy*, D. L. Sparks, Ed., vol. 132, Academic Press, Jan. 1, 2015, pp. 139–159. DOI: 10.1016/bs.agron.2015.02.002 (cit. on p. 6).

[17] C. Gomez, R. A. Viscarra Rossel, and A. B. McBratney, “Soil organic carbon prediction by hyperspectral remote sensing and field vis-NIR spectroscopy: An australian case study,” *Geoderma*, vol. 146, no. 3, pp. 403–411, Aug. 31, 2008, ISSN: 0016-7061. DOI: 10.1016/j.geoderma.2008.06.011 (cit. on p. 6).

[18] A. M. Mouazen, M. R. Maleki, J. De Baerdemaeker, and H. Ramon, “On-line measurement of some selected soil properties using a VIS–NIR sensor,” *Soil and Tillage Research*, vol. 93, no. 1, pp. 13–27, Mar. 1, 2007, ISSN: 0167-1987. DOI: 10.1016/j.still.2006.03.009 (cit. on p. 6).

[19] X. Zhang, N. H. Younan, and C. G. O’Hara, “Wavelet domain statistical hyperspectral soil texture classification,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, no. 3, pp. 615–618, Mar. 2005, ISSN: 0196-2892. DOI: 10.1109/TGRS.2004.841476 (cit. on p. 6).

[20] N. M. Dhawale, V. I. Adamchuk, S. O. Prasher, R. A. V. Rossel, A. A. Ismail, and J. Kaur, “Proximal soil sensing of soil texture and organic matter with a prototype portable mid-infrared spectrometer,” *European Journal of Soil Science*, vol. 66, no. 4, pp. 661–669, Jul. 1, 2015, ISSN: 1365-2389. DOI: 10.1111/ejss.12265 (cit. on p. 6).

[21] E. U. Hobley and I. Prater, “Estimating soil texture from vis–NIR spectra,” *European journal of soil science*, vol. 70, no. 1, pp. 83–95, 2019, Publisher: Wiley Online Library (cit. on p. 6).

[22] B. Sudarsan, W. Ji, V. Adamchuk, and A. Biswas, “Characterizing soil particle sizes using wavelet analysis of microscope images,” *Computers and Electronics in Agriculture*, vol. 148, pp. 217–225, May 2018, ISSN: 01681699. DOI: 10.1016/j.compag.2018.03.019 (cit. on pp. 7, 10, 44).

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015, ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-015-0816-y (cit. on pp. 8, 11, 12).

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105 (cit. on pp. 8, 44).

[25] R. Hryciw and S. Raschke, “Development of computer vision technique for in situ soil characterization,” *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1526, pp. 86–97, Jan. 1, 1996, ISSN: 0361-1981. DOI: 10.3141/1526-11 (cit. on p. 10).

[26] P. Breul and R. Gourves, “In field soil characterization: Approach based on texture image analysis,” *Journal of geotechnical and geoenvironmental engineering*, vol. 132, no. 1, pp. 102–107, 2006, Publisher: American Society of Civil Engineers (cit. on p. 10).

[27] L. Qi, V. Adamchuk, H.-H. Huang, M. Leclerc, Y. Jiang, and A. Biswas, “Proximal sensing of soil particle sizes using a microscope-based sensor and bag of visual words model,” *Geoderma*, vol. 351, pp. 144–152, 2019 (cit. on p. 10).

[28] R. K. Swetha, P. Bende, K. Singh, S. Gorthi, A. Biswas, B. Li, D. C. Weindorf, and S. Chakraborty, “Predicting soil texture from smartphone-captured digital images and an application,” *Geoderma*, vol. 376, p. 114 562, 2020, Publisher: Elsevier (cit. on p. 10).

[29] C. J. Moran, “Image processing and soil micromorphology,” in *Developments in Soil Science*, vol. 22, Elsevier, 1993, pp. 459–482 (cit. on p. 11).

[30] A. Sofou, G. Evangelopoulos, and P. Maragos, “Soil image segmentation and texture analysis: A computer vision approach,” *IEEE Geoscience and Remote Sensing Letters*, vol. 2, no. 4, pp. 394–398, Oct. 2005, ISSN: 1545-598X. DOI: 10.1109/LGRS.2005.851752 (cit. on p. 11).

[31] V. Marcelino, V. Cnudde, S. Vansteelandt, and F. Caro, “An evaluation of 2d-image analysis techniques for measuring soil microporosity,” *European Journal of Soil Science*, vol. 58, no. 1, pp. 133–140, 2007, Publisher: Wiley Online Library (cit. on p. 11).

[32] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015, Publisher: Nature Publishing Group (cit. on pp. 11, 14).

[33] S. Fernández, A. Graves, and J. Schmidhuber, “An application of recurrent neural networks to discriminative keyword spotting,” in *Proceedings of the 17th international conference on Artificial neural networks*, ser. ICANN’07, Berlin, Heidelberg: Springer-Verlag, Sep. 9, 2007, pp. 220–229, ISBN: 978-3-540-74693-5 (cit. on p. 11).

[34] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018, Publisher: IEEE (cit. on p. 12).

[35] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, “A survey on deep learning in medical image analysis,” *Medical image analysis*, vol. 42, pp. 60–88, 2017, Publisher: Elsevier (cit. on p. 12).

[36] G. B. Goh, N. O. Hodas, and A. Vishnu, “Deep learning for computational chemistry,” *Journal of computational chemistry*, vol. 38, no. 16, pp. 1291–1307, 2017, Publisher: Wiley Online Library (cit. on p. 12).

[37] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle, “Deep learning for computational biology,” *Molecular systems biology*, vol. 12, no. 7, p. 878, 2016 (cit. on p. 12).

[38] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-Y. Chang, and T. Sainath, “Deep learning for audio signal processing,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019, Publisher: IEEE (cit. on p. 12).

[39] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, “A review on deep learning techniques applied to semantic segmentation,” *arXiv preprint arXiv:1704.06857*, 2017 (cit. on p. 12).

[40] Y. Rivenson, Z. Göröcs, H. Günaydin, Y. Zhang, H. Wang, and A. Ozcan, “Deep learning microscopy,” *Optica*, vol. 4, no. 11, pp. 1437–1443, 2017, Publisher: Optical Society of America (cit. on p. 12).

[41] A. Kamilaris and F. X. Prenafeta-Boldú, “Deep learning in agriculture: A survey,” *Computers and electronics in agriculture*, vol. 147, pp. 70–90, 2018, Publisher: Elsevier (cit. on p. 12).

[42] M. M. Ghazi, B. Yanikoglu, and E. Aptoula, “Plant identification using deep neural networks via optimization of transfer learning parameters,” *Neurocomputing*, vol. 235, pp. 228–235, 2017, Publisher: Elsevier (cit. on p. 12).

[43] M. Rahnemoonfar and C. Sheppard, “Deep count: Fruit counting based on deep simulated learning,” *Sensors*, vol. 17, no. 4, p. 905, 2017, Publisher: Multidisciplinary Digital Publishing Institute (cit. on p. 12).

[44] A. K. Mortensen, M. Dyrmann, H. Karstoft, R. N. Jørgensen, and R. Gislum, “Semantic segmentation of mixed crops using deep convolutional neural network,” in *Proc. of the International Conf. of Agricultural Engineering (CIGR)*, 2016 (cit. on p. 12).

[45] A. Milioto, P. Lottes, and C. Stachniss, “Real-time blob-wise sugar beets vs weeds classification for monitoring fields using convolutional neural networks,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 4, p. 41, 2017, Publisher: Copernicus GmbH (cit. on p. 12).

[46] T. Behrens, K. Schmidt, R. A. MacMillan, and R. A. V. Rossel, “Multi-scale digital soil mapping with deep learning,” *Scientific reports*, vol. 8, no. 1, pp. 1–9, 2018, Publisher: Nature Publishing Group (cit. on p. 12).

[47] J. Padarian, B. Minasny, and A. B. McBratney, “Using deep learning for digital soil mapping,” *Soil*, vol. 5, no. 1, pp. 79–89, 2019, Publisher: Copernicus GmbH (cit. on p. 12).

[48] K. Maladkar. (Jan. 25, 2018). Overview of convolutional neural network in image classification, Analytics India Magazine, [Online]. Available: https://analyticsindiamag.com/convolutional-neural-network-image-classification-overview/ (visited on 08/28/2020) (cit. on p. 12).

[49] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of solid-state circuits*, vol. 23, no. 2, pp. 358–367, 1988, Publisher: IEEE (cit. on p. 12).

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016 (cit. on pp. 13, 14).

[51] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998 (cit. on pp. 13, 15, 16).

[52] R. Theart. (Nov. 29, 2017). Getting started with PyTorch for deep learning (part 3: Neural network basics), Code to Light, [Online]. Available: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/ (visited on 08/24/2020) (cit. on p. 13).

[53] Y. LeCun, C. Cortes, and C. Burges. (). MNIST handwritten digit database. dataset, [Online]. Available: http://yann.lecun.com/exdb/mnist/ (visited on 03/19/2019) (cit. on p. 14).

[54] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989, Publisher: MIT Press (cit. on pp. 14, 16).

[55] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 437–478 (cit. on pp. 15, 62).

[56] G. E. Hinton, "A practical guide to training restricted boltzmann machines," in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 599–619 (cit. on p. 15).

[57] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014 (cit. on pp. 15, 47).

[58] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*. 2017, _eprint: 1711.05101 (cit. on pp. 15, 47).

[59] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The marginal value of adaptive gradient methods in machine learning,” in *Advances in neural information processing systems*, 2017, pp. 4148–4158 (cit. on p. 15).

[60] S. Gugger. (Jul. 2, 2018). AdamW and super-convergence is now the fastest way to train neural nets, [Online]. Available: https://www.fast.ai/2018/07/02/adam-weight-decay/ (cit. on p. 15).

[61] W. Samek, T. Wiegand, and K.-R. Müller, “Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models,” *arXiv preprint arXiv:1708.08296*, 2017 (cit. on p. 15).

[62] A. Singh, S. Sengupta, and V. Lakshminarayanan, “Explainable deep learning models in medical image analysis,” *arXiv preprint arXiv:2005.13799*, 2020 (cit. on p. 16).

[63] D. Krishnan, T. Tay, and R. Fergus, “Blind deconvolution using a normalized sparsity measure,” in *CVPR 2011*, Jun. 2011, pp. 233–240. DOI: 10.1109/CVPR.2011.5995521 (cit. on p. 16).

[64] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014 (cit. on pp. 16, 17).

[65] G. Touloupas, A. Lauber, J. Henneberger, A. Beck, and A. Lucchi, “A convolutional neural network for classifying cloud particles recorded by imaging probes,” *Atmospheric Measurement Techniques*, vol. 13, no. 5, pp. 2219–2239, 2020. DOI: 10.5194/amt-13-2219-2020 (cit. on p. 17).

[66] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and*

Knowledge-Based Systems, vol. 6, no. 2, pp. 107–116, 1998, Publisher: World Scientific (cit. on p. 17).

[67] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778 (cit. on p. 17).

[68] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, ISBN: 9781538664209 Publisher: IEEE. DOI: 10.1109/cvpr.2018.00474 (cit. on p. 18).

[69] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019 (cit. on pp. 18, 61).

[70] M. Cimpoi, S. Maji, I. Kokkinos, and A. Vedaldi, "Deep filter banks for texture recognition, description, and segmentation," *International Journal of Computer Vision*, vol. 118, no. 1, pp. 65–94, May 1, 2016, ISSN: 1573-1405. DOI: 10.1007/s11263-015-0872-3 (cit. on pp. 18, 60).

[71] V. Andrearczyk and P. F. Whelan, "Using filter banks in convolutional neural networks for texture classification," *Pattern Recognition Letters*, vol. 84, pp. 63–69, Dec. 1, 2016, ISSN: 0167-8655. DOI: 10.1016/j.patrec.2016.08.016 (cit. on p. 18).

[72] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034 (cit. on p. 22).

[73] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *2016 eighth international conference on quality of multimedia experience (QoMEX)*, IEEE, 2016, pp. 1–6 (cit. on p. 23).

[74] AnMo Electronics Corporation. (2018). Dino-lite microscope, [Online]. Available: https://www.dino-lite.com/products_detail.php?index_m1_id=9&index_m2_id=42&index_id=139 (visited on 08/28/2020) (cit. on p. 25).

[75] V. Adamchuk, A. Biswas, L. Qi, M. Leclerc, B. Sudarsan, and W. Ji, “Apparatus for analyzing a sample of granular material,” pat. 10 495 568, Dec. 3, 2019 (cit. on pp. 25, 26).

[76] R. Agarwal, “Edge detection in images using modified bit-planes sobel operator,” in *Proceedings of the Third International Conference on Soft Computing for Problem Solving*, M. Pant, K. Deep, A. Nagar, and J. C. Bansal, Eds., ser. Advances in Intelligent Systems and Computing, New Delhi: Springer India, 2014, pp. 203–210, ISBN: 978-81-322-1771-8. DOI: 10.1007/978-81-322-1771-8_18 (cit. on p. 28).

[77] H. Mir, P. Xu, and P. v. Beek, “An extensive empirical evaluation of focus measures for digital photography,” in *Digital Photography X*, vol. 9023, International Society for Optics and Photonics, Mar. 7, 2014, p. 90230I. DOI: 10.1117/12.2042350 (cit. on p. 28).

[78] A. Wise. (Jan. 12, 2013). Automatic exposure bracketing (AEB) explained, Alex Wise Photography, [Online]. Available: https://www.alexwisephotography.net/blog/2013/01/12/automatic-exposure-bracketing-aeb-explained/ (visited on 08/27/2020) (cit. on p. 29).

[79] J. Dickman, *Perfect digital photography*. New York: McGraw-Hill, 2009, ISBN: 978-0-07-160166-5 (cit. on p. 29).

[80] E. Reinhard, W. Heidrich, P. Debevec, S. Pattanaik, G. Ward, and K. Myszkowski, *High dynamic range imaging: acquisition, display, and image-based lighting*. Morgan Kaufmann, 2010 (cit. on p. 29).

[81] E. Hecht, *Optics*, Fifth edition. Boston: Pearson Education, Inc., 2017, ISBN: 978-1-292-09693-3 (cit. on p. 30).

[82] D. Johnston, *Random number generators - principles and practices : a guide for engineers and programmers*. Berlin Boston: Walter de Gruyter GmbH, 2018, ISBN: 978-1-5015-1513-2 (cit. on p. 38).

[83] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in neural information processing systems*, 2019, pp. 8026–8037 (cit. on p. 41).

[84] R. A. Adams and J. J. Fournier, *Sobolev spaces*. Academic Press, 2003, ISBN: 978-0-12-044143-3 (cit. on pp. 42, 43).

[85] C. Sammut and G. I. Webb, *Encyclopedia of machine learning*. Springer Science & Business Media, 2011 (cit. on p. 43).

[86] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in PyTorch,” 2017 (cit. on p. 46).

[87] N. R. Draper and H. Smith, *Applied regression analysis*, 3rd ed., 1 online resource (xvii, 706 pages) : illustrations vols., Wiley series in probability and statistics. Texts and references section. New York: Wiley, 1998, ISBN: 978-0-471-17082-2 (cit. on p. 50).

[88] Y. S. Aurelio, G. M. de Almeida, C. L. de Castro, and A. P. Braga, “Learning from imbalanced data sets with weighted cross-entropy function,” *Neural Processing Letters*, vol. 50, no. 2, pp. 1937–1949, 2019, Publisher: Springer (cit. on p. 58).

[89] S. Gugger. (Apr. 7, 2018). The 1cycle policy, [Online]. Available: https://sgugger.github.io/the-1cycle-policy.html (cit. on p. 62).

[90] L. N. Smith, “A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay,” *arXiv preprint arXiv:1803.09820*, 2018 (cit. on p. 62).

[91] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012, Publisher: JMLR. org (cit. on p. 63).

[92] O. Tange, “GNU parallel 20200922 (’ginsburg’),” Sep. 22, 2020. DOI: 10.5281/zenodo.4045386.

Appendix A. Particle Size Taxonomies

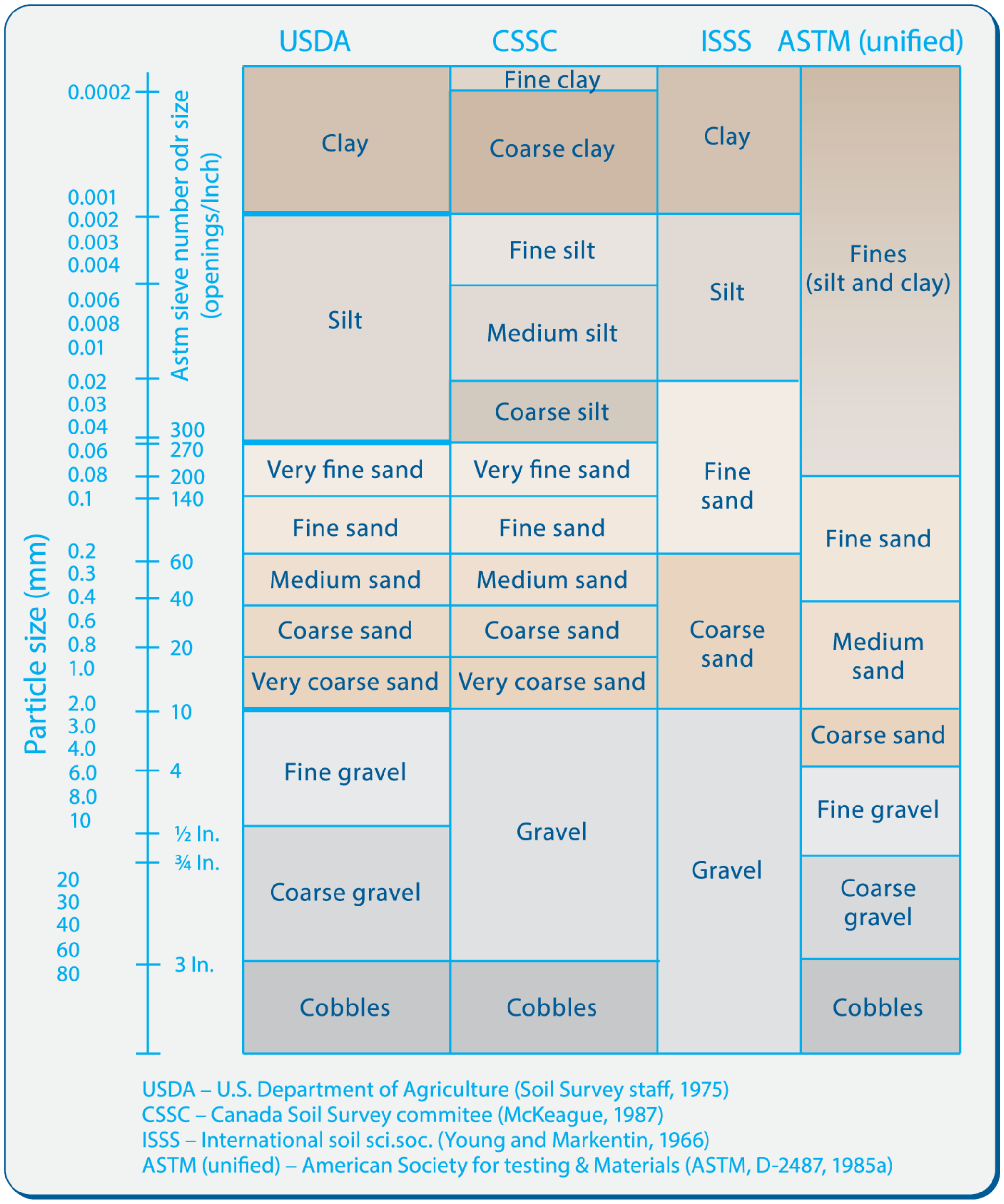


Figure A.1: Particle Size Classification Systems [9]

Appendix B. Image Acquisition Protocol

B.1 Initial Hardware Setup

Removing the digital microscope from the microscope holder

1. Remove the microscope holder cap by unscrewing it.
2. Disconnect the microscope from its USB cable.
3. Loosen the plastic screw on the side of the microscope holder sufficiently to allow the microscope to slide out of the holder.

Setting the digital microscope magnification and polarization

4. Roll the black dial on the side of the microscope as far as it will go to achieve maximum magnification (approximately $220\times$).
5. Secure the magnification lock (small notched switch next to the magnification dial) in place and make sure the magnification dial can no longer be turned.
6. Rotate the polarization filter (toothed black dial near microscope lens) as far as it will go towards the '+' setting.

Securing the digital microscope in place

7. Loosen the plastic screw on the side of the microscope holder sufficiently to allow the microscope to slide into the holder.
8. While holding the microscope holder on an angle with the opening for the magnification dial facing down, allow the microscope to slide down along the inner

wall of the holder until the microscope's toothed black dial comes to a rest on the small inner lip inside the holder (above the opening for the magnification dial), as in Figure 1 [*not shown*].

9. While keeping the microscope snug to the inner lip, rotate it as necessary to align the USB port with the rectangular window underneath the holder, as in Figure 2 [*not shown*].

10. Once the microscope is both resting on the inner lip and aligned with the USB window, tighten the plastic crew on the side of the holder to secure the microscope in place, taking care not to overtighten and risk damage to the microscope.

11. Connect the USB cable to the microscope via the USB window.

Attaching the microscope holder cap

12. Ensure the glass window in the cap of the microscope holder is clean on both sides. It can be cleaned using dish soap or lens cleaning solution and wiped dry with a microfiber cloth.

13. Place a single metal shim (O-Ring) on the upper lip of the base of the microscope holder.

14. Screw the cap into place, such that reasonably tight contact is made between the base, the shim, and the cap.

B.2 Calibration

Instrument Setup

15. Ensure that the microscope is secured and the cap is attached according to the instructions of the previous section.

16. Clean the calibration target using dish soap, lens cleaner, or the like.

17. Place the calibration target in the calibration target harness, ensuring that the side of the calibration target on which the crosshair pattern is engraved faces towards you, such that it will be placed in direct contact with the window of the microscope holder. Apply pressure along the edges of the calibration target to attach it securely to the harness.

18. Place the calibration target harness on the microscope holder, applying light pressure around its perimeter to ensure contact between the calibration target and the window of the microscope holder.

jFocus Software Setup

19. Install the DNVideoX SDK by unzipping “DNVideoX_V3.0.43_Installer.zip” to a directory, then right-clicking on the installer and selecting “Run as Administrator”.

20. Save the jFocus executable, “jFocus.exe”, to the directory of your choice.

Rough Adjustment of the Focal Length

21. Open the jFocus application by executing jFocus.exe.

22. The Dino-Lite microscope should be selected by default, and the preview window should display a video stream as captured by the microscope. If the preview window is black, make sure to close any DinoCapture sessions or other jFocus sessions that may be running, and hit the “Refresh” button.

23. Starting with the microscope holder cap screwed fully to the base, slowly unscrew the cap to increase the focal length until the image in the preview window becomes relatively sharp. The origin (central crosshairs) of the calibration target should appear in the centre of the preview window, along with at least 5 measurement ticks on each side.

Saving the Calibration Image

24. Enter “C” in the Sample ID text box, then select the “Save PNG Image” button. A message box should appear indicating the name of a PNG image that was saved to the Desktop. If the image file could not be saved, an error message may appear.

B.3 Capturing Images of Soil Samples

Cleaning

25. Make sure the window of the microscope holder is clean on both sides. For best results, use dish soap or a lens cleaning solution, rinse and wipe dry using a microfiber cloth.

Calibration

26. A new calibration image should be taken at the beginning of each day in accordance with the instructions provided above.

Images of Soil Samples

27. Evenly spread a 20g sample of the soil under analysis over the window of the microscope holder.

28. Enter a unique identifier for the soil sample in the Sample ID text box.

29. Select the “Save PNG Image” button. A message box will indicate the filename of the PNG image on the Desktop.

Between Soil Samples

30. Be sure to clean the microscope window again before proceeding to capturing images of a new soil sample.

Appendix C. Soil Sample Images

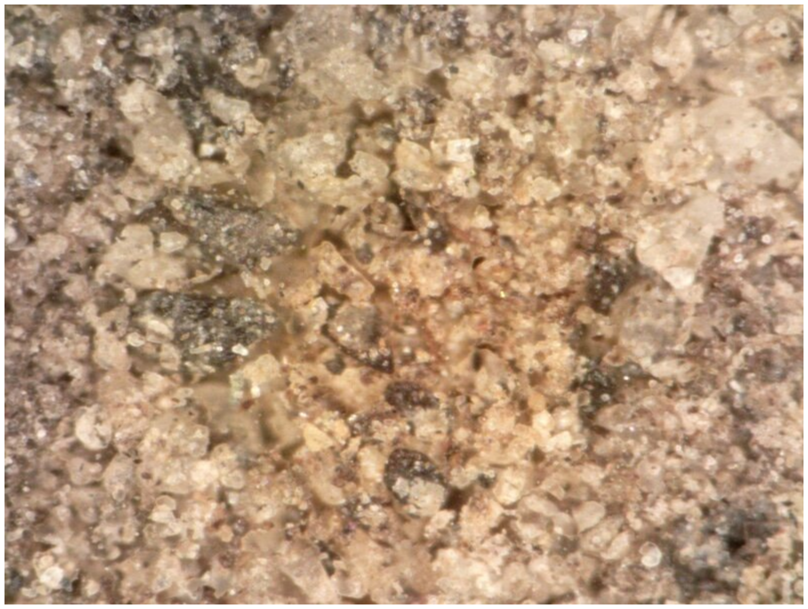


Figure C.1: $(sand_4\ silt_4\ clay_4\ OM) = (0.819\ 0.112\ 0.059\ 0.010)$

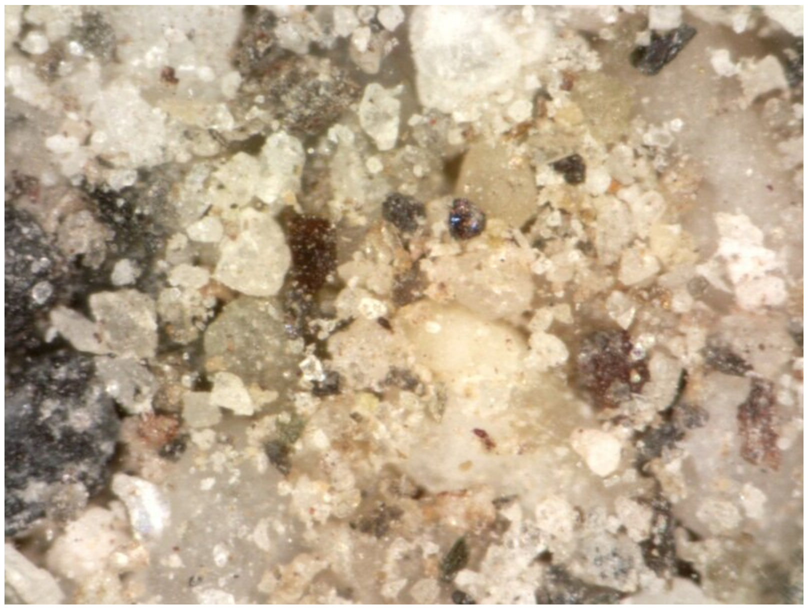


Figure C.2: $(sand_4\ silt_4\ clay_4\ OM) = (0.945\ 0.013\ 0.040\ 0.002)$

Appendix D. AL2019 Index Format

key	image filename	`jFocus12_11660_20190527145338_edr.png`
STSAND	sand content (three-way)	0.687
STSILT	silt content (three-way)	0.206
STCLAY	sand content (three-way)	0.107
SAND	sand content (four-way)	0.663642
SILT	silt content (four-way)	0.198996
CLAY	clay content (four-way)	0.103362
OM	organic matter content (four-way)	0.034
STCLASS	USDA soil texture class (1 to 12)	9
image-height	image height in pixels	1920
image-width	image width in pixels	2560
qrank	quality percentile	0.97826
sample	unique sample ID	`11421`
suspicions	list of suspicions about the image	`[poor quality]`
utc-timestamp	image capture date and time	`20190527145338`
ftp-modtime	FTP modification time	`20190527122154`

Table D.1: Partial listing of AL2019 index fields with descriptions and sample values

Appendix E. Computing Resources

E.1 Self-Hosted Computers

E.1.1 Image Acquisition Computer

Images were captured via the jFocus software on a notebook computer at A&L Canada Laboratories running the Windows 10 OS and connected to the microscope via a USB 3.0 cable.

E.1.2 Model Training and Testing Computer

CPU	`Intel i5-4590T 3.0GHz, 16GB RAM`
GPU	`NVIDIA GeForce GTX 1050Ti, 4GB`
OS	`Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-112-generic x86_64)`

Table E.1: Model Training and Testing Computer (`euclid`)

E.1.3 Development Computers

All software, apart from jFocus, was developed on a MacBook computer with the specs shown in Table E.2. The jFocus software was developed in Microsoft Visual Studio on a Windows 10 PC in the PASS Lab. Remote connections were initiated from the development computers as necessary to interact with the Linux server (`euclid`), GitLab servers, PASS FTP server, and Wasabi S3 servers.

Model	`MacBook Pro (Retina, 13-inch, Mid 2014)`
CPU	`2.6GHz Dual-Core Intel Core i5`
Memory	`8GB 1600MHz DDR3`
Graphics	`Intel Iris 1536MB`
OS	`MacOS versions 10.14 to 10.15.6`

Table E.2: MacOS Development Computer

E.2 Servers and Cloud Computing Resources

E.2.1 FTP Server

The PASS Lab FTP server was a basic single-drive Western Digital Live NAS device.

E.2.2 S3 Cloud Storage

The original files uploaded from A&L Canada to the FTP server and the AL2019 dataset were stored in the cloud using S3 object storage provided by Wasabi.

E.2.3 GitLab

The jFocus, AL2019Mirror, AL2019Packager, and DeepSoil software repositories were all hosted on GitLab. See Appendix G for hyperlinks.

Appendix F.   Virtual Environment

The virtual environment in which the dataset preprocessing, model training, and model testing were performed was defined as a Docker container using a Dockerfile which is available for download from the DeepSoil software repository.

Within the virtual environment of the Docker container, the `conda` package manager was installed and used to configure the Python environment in which the dataset preprocessing, model training, model testing, and statistical evaluation components were executed. Executing the command `conda env export` within this environment lists all of the package and library versions installed. The output of that command is listed here for reference.

```
name: base
channels:
  - pytorch
  - conda-forge
  - defaults
dependencies:
  - _libgcc_mutex=0.1=main
  - albumentations=0.4.5=py_0
  - argcomplete=1.11.1=py_1
  - asciitree=0.3.3=py_2
  - asn1crypto=1.3.0=py37_0
  - blas=1.0=mkl
  - bokeh=2.1.1=py37_0
  - boto3=1.9.66=py37_0
  - botocore=1.12.189=py_0
  - bzip2=1.0.8=h7b6447c_0
  - ca-certificates=2020.6.24=0
  - cairo=1.14.12=h8948797_3
  - certifi=2020.6.20=py37_0
  - cffi=1.14.0=py37h2e261b9_0
  - chardet=3.0.4=py37_1003
  - click=7.1.2=py_0
  - cloudpickle=1.4.1=py_0
  - conda=4.8.3=py37_0
  - conda-package-handling=1.6.0=py37h7b6447c_0
  - cryptography=2.8=py37h1ba5d50_0
  - cudatoolkit=10.1.243=h6bb024c_0
  - cycler=0.10.0=py37_0
  - cytoolz=0.10.1=py37h7b6447c_0
  - dask=2.19.0=py_0
  - dask-core=2.19.0=py_0
  - dbus=1.13.16=hb2f20db_0
  - decorator=4.4.2=py_0
  - distributed=2.19.0=py37_0
  - docutils=0.16=py37_1
  - et_xmlfile=1.0.1=py37_0
  - expat=2.2.9=he6710b0_2
  - fasteners=0.15=py_0
  - ffmpeg=4.0=hcdf2ecd_0
  - fontconfig=2.13.0=h9420a91_0
  - freeglut=3.0.0=hf484d3e_5
  - freetype=2.10.2=h5ab3b9f_0
  - fsspec=0.7.4=py_0
  - geos=3.8.0=he6710b0_0
  - git=2.23.0=pl526hacde149_0
  - gitdb=4.0.5=py_0
  - gitpython=3.1.3=py_1
  - glib=2.63.1=h5a9c865_0
  - graphite2=1.3.14=h23475e2_0
  - gst-plugins-base=1.14.0=hbbd80ab_1
  - gstreamer=1.14.0=hb453b48_1
  - harfbuzz=1.8.8=hffaf4a1_0
  - hdf5=1.10.2=hba1933b_1
  - heapdict=1.0.1=py_0
  - icu=58.2=he6710b0_3
  - idna=2.8=py37_0
  - imageio=2.8.0=py_0
  - imgaug=0.4.0=py_1
- importlib-metadata=1.7.0=py37_0
- importlib_metadata=1.7.0=0
- intel-openmp=2020.1=217
- jasper=2.0.14=h07fcdf6_1
- jdcal=1.4.1=py_0
- jinja2=2.11.2=py_0
- jmespath=0.9.4=py_0
- jpeg=9b=h024ee3a_2
- kiwisolver=1.2.0=py37hfd86e86_0
- kornia=0.3.0=pyh9f0ad1d_0
- krb5=1.17.1=h173b8e3_0
- libcurl=7.68.0=h20c2e04_0
- libedit=3.1.20181209=hc058e9b_0
- libffi=3.2.1=hd88cf55_4
- libgcc-ng=9.1.0=hdf63c60_0
- libgfortran-ng=7.3.0=hdf63c60_0
- libglu=9.0.0=hf484d3e_1
- libmagic=5.39=hed695b0_0
- libopencv=3.4.2=hb342d67_1
- libopus=1.3.1=h7b6447c_0
- libpng=1.6.37=hbc83047_0
- libssh2=1.9.0=h1ba5d50_1
- libstdcxx-ng=9.1.0=hdf63c60_0
- libtiff=4.1.0=h2733197_0
- libuuid=1.0.3=h1bed415_2
- libvpx=1.7.0=h439df22_0
- libxcb=1.14=h7b6447c_0
- libxml2=2.9.9=hea5a465_1
- locket=0.2.0=py37_1
- markupsafe=1.1.1=py37h7b6447c_0
- matplotlib=3.1.3=py37_0
- matplotlib-base=3.1.3=py37hef1b27d_0
- mkl=2020.1=217
- mkl-service=2.3.0=py37he904b0f_0
- mkl_fft=1.1.0=py37h23d657b_0
- mkl_random=1.1.1=py37h0573a6f_0
- monotonic=1.5=py_0
- msgpack-python=1.0.0=py37hfd86e86_1
- ncurses=6.1=he6710b0_1
- networkx=2.4=py_0
- ninja=1.9.0=py37hfd86e86_0
- numcodecs=0.6.4=py37he6710b0_0
- numpy=1.18.5=py37ha1c710e_0
- numpy-base=1.18.5=py37hde5b4d6_0
- olefile=0.46=py37_0
- opencv=3.4.2=py37h6fd60c2_1
- openpyxl=3.0.3=py_0
- openssl=1.1.1g=h7b6447c_0
- packaging=20.4=py_0
- pandas=1.0.5=py37h0573a6f_0
- partd=1.1.0=py_0
- pcre=8.44=he6710b0_0
- perl=5.26.2=h14c3975_0
- pillow=7.1.2=py37hb39fc2d_0
- pip=20.1.1=py37_1
- pixman=0.40.0=h7b6447c_0
- psutil=5.7.0=py37h7b6447c_0
- py-opencv=3.4.2=py37hb342d67_1
- pycosat=0.6.3=py37h7b6447c_0
- pycparser=2.19=py37_0
- pyopenssl=19.1.0=py37_0
- pyparsing=2.4.7=py_0
- pyqt=5.9.2=py37h05f1152_2
- pysocks=1.7.1=py37_0
- python=3.7.4=h265db76_1
- python-dateutil=2.8.1=py_0
- python-magic=0.4.15=py37hc8dfbb8_1002
- python-ternary=1.0.7=pyh9f0ad1d_0
- python_abi=3.7=1_cp37m
- pytorch=1.5.1=py3.7_cuda10.1.243_cudnn7.6.3_0
- pytz=2020.1=py_0
- pywavelets=1.1.1=py37h7b6447c_0
- pyyaml=5.3.1=py37h7b6447c_0
- qt=5.9.7=h5867ecd_1
- readline=7.0=h7b6447c_5
- requests=2.22.0=py37_1
- ruamel_yaml=0.15.87=py37h7b6447c_0
- s3transfer=0.1.13=py37_0
- scikit-image=0.16.2=py37h0573a6f_0
- scipy=1.5.0=py37h0b6359f_0
- setuptools=45.2.0=py37_0
- shapely=1.7.0=py37h98ec03d_0
- sip=4.19.8=py37hf484d3e_0
- six=1.14.0=py37_0
- smmap=3.0.2=py_0
- sortedcontainers=2.2.2=py_0
- sqlite=3.31.1=h7b6447c_0
- tblib=1.6.0=py_0
- tini=0.18.0=h7b6447c_0
- tk=8.6.8=hbc83047_0
- toolz=0.10.0=py_0
- torchvision=0.6.1=py37_cu101
- tornado=6.0.4=py37h7b6447c_1
- tqdm=4.47.0=py_0
- typing_extensions=3.7.4.2=py_0
- unzip=6.0=h611a1e1_0
- urllib3=1.25.8=py37_0
- wheel=0.34.2=py37_0
- xz=5.2.4=h14c3975_4
- yaml=0.1.7=had09818_2
- zarr=2.3.2=py_0
- zict=2.0.0=py_0
- zipp=3.1.0=py_0
- zlib=1.2.11=h7b6447c_3
- zstd=1.3.7=h0b5b093_0
- pip:
  - master-sake==1.2
```

Appendix G. Source Code

G.1 jFocus

The source code of the jFocus microscope control software is available on GitLab at the following URL:

https://gitlab.com/k8spiers/jFocus

G.2 AL2019Mirror

The source code of the Al2019Mirror microscope control software is available on GitLab at the following URL:

https://gitlab.com/k8spiers/al2019mirror

G.3 AL2019Packager

The source code of the Al2019Packager microscope control software is available on GitLab at the following URL:

https://gitlab.com/k8spiers/al2019packager

G.4 DeepSoil

The source code for the dataset preprocessing, model training, model testing, and statistical evaluation components is available on GitLab at the following URL:

https://gitlab.com/k8spiers/deepsoil

Appendix H. Excerpt from Presets YAML File

Presets for settings governing the dataset preprocessing, model training, and model testing components were stored in a YAML file. Below is a very small excerpt. The preset definitions are recursive, inheriting from parent definitions via the `extends` keyword.

```
default:
  batchsize: null
  device: null
  forbidden: [suspicions]
  indexpath: data/al2019/index.yml
  splits: 10
  keys: [STCLASS]
  learningrate: 0.001
  maxepochs: null
  maxtime: 300
  momentum: 0.9
  netclass: LeNetMod
  normalize: true
  numpytransforms: []
  optimization: SGD
  qrankpercent: null
  schedule: plateau
  style: classify
  tensortransforms: []
  traintestratio: 0.8
  verbosity: 0

al2019_1600_fft1remainder_threeway:
  batchsize: 23
  extends: al2019_fft1remainder_threeway
  learningrate: 0.001
  maxtime: 4800
  netclass: FFT1Remainder1600
  numpytransforms:
    - "RandomCrop:1600,1600"
    - HorizontalFlip
    - RandomRotate90

al2019_2500_fft1remainder_threeway:
  batchsize: 15
  extends: al2019_fft1remainder_threeway
  learningrate: 0.001
  maxtime: 4800
  netclass: FFT1Remainder2500
  numpytransforms:
    - "RandomCrop:1600,2500"
    - HorizontalFlip

al2019_400_fft1remainder_threeway:
  batchsize: "/2"
  extends: al2019_fft1remainder_threeway
  maxtime: 1000
  netclass: FFT1Remainder400
  numpytransforms:
    - "SmallestMaxSize:480"
    - "RandomCrop:400,400"
    - HorizontalFlip
    - RandomRotate90

al2019_STCLASS_resnet18:
  batchsize: 50
  extends: al2019_STCLASS_lenetmod
  learningrate: 0.01
  momentum: 0.8
  netclass: ResNet18
  numpytransforms:
    - "RandomCrop:300,300"
    - HorizontalFlip
    - RandomRotate90

al2019_threeway_resnet18_600_300:
  extends: al2019_STSAND/(STSAND+STSILT)_resnet18_600
  keys: [STSAND, STSILT, STCLAY]
  maxtime: 3600

al2019_threeway_resnet50_600_300:
  extends: al2019_threeway_resnet18_600_300
  netclass: ResNet50
  batchsize: 20
  learningrate: 0.01
  maxtime: 7200

al2019_threeway_mobilenetv2_600_300:
  extends: al2019_threeway_resnet18_600_300
  netclass: MobileNetV2
  batchsize: 25
  learningrate: 0.01
  maxtime: 7200

al2019_fourway_mobilenetv2_600_300:
  extends: al2019_threeway_mobilenetv2_600_300
  keys: [CLAY, SAND, SILT, OM]

al2019_threeway_mobilenetv2_400_200:
  extends: al2019_threeway_mobilenetv2_600_300
  numpytransforms:
    - "SmallestMaxSize:400"
    - "RandomCrop:200,200"
    - HorizontalFlip
    - RandomRotate90
  batchsize: 65
  splits: 3

al2019_fourway_mobilenetv2_400_200:
  extends: al2019_threeway_mobilenetv2_400_200
  keys: [CLAY, SAND, SILT, OM]
```

```
al2019_threeway_mobilenetv2_300_300:
  extends: al2019_threeway_mobilenetv2_600_300
  numpytransforms:
    - "SmallestMaxSize:300"
    - "RandomCrop:300,300"
    - HorizontalFlip
    - RandomRotate90

al2019_threeway_mobilenetv2remainder_300_300:
  extends: al2019_threeway_mobilenetv2_300_300
  netclass: MobileNetV2Remainder

al2019_fourway_mobilenetv2_300_300:
  extends: al2019_threeway_mobilenetv2_300_300
  keys: [CLAY, SAND, SILT, OM]

al2019_fourway_mobilenetv2remainder_300_300:
  extends: al2019_fourway_mobilenetv2_300_300
  netclass: MobileNetV2Remainder

al2019_STCLASS_mobilenetv2_300_300:
  extends: al2019_threeway_mobilenetv2_300_300
  keys: [STCLASS]
  style: classify
  splits: 3

al2019_threeway_mobilenetv2_300_280:
  extends: al2019_threeway_mobilenetv2_600_300
  numpytransforms:
    - "SmallestMaxSize:300"
    - "RandomCrop:280,280"
    - HorizontalFlip
    - RandomRotate90

al2019_fourway_mobilenetv2_300_280:
  extends: al2019_threeway_mobilenetv2_300_280
  keys: [CLAY, SAND, SILT, OM]

al2019_threeway_mobilenetv2_400_400:
  extends: al2019_threeway_mobilenetv2_600_300
  numpytransforms:
    - "SmallestMaxSize:400"
    - "RandomCrop:400,400"
    - HorizontalFlip
    - RandomRotate90
  batchsize: 14

al2019_fourway_mobilenetv2_400_400:
  extends: al2019_threeway_mobilenetv2_400_400
  keys: [CLAY, SAND, SILT, OM]

al2019_threeway_resnet18_600_400:
  extends: al2019_threeway_resnet18_600_300
  batchsize: 28
  numpytransforms:
    - "SmallestMaxSize:600"
    - "RandomCrop:400,400"
    - HorizontalFlip
    - RandomRotate90

al2019_threeway_resnet50_600_400:
  extends: al2019_threeway_resnet18_600_400
  netclass: ResNet50
  batchsize: 11
  learningrate: 0.01
  maxtime: 7200

al2019_threeway_mobilenetv2_600_400:
  extends: al2019_threeway_resnet18_600_400
  netclass: MobileNetV2
  batchsize: 14
  learningrate: 0.01
  maxtime: 7200

al2019_fourway_mobilenetv2_600_400:
  extends: al2019_threeway_mobilenetv2_600_400
  keys: [CLAY, SAND, SILT, OM]

al2019_threeway_squeezenet_600_400:
  extends: al2019_threeway_resnet18_600_400
  netclass: SqueezeNet1_1
  learningrate: 0.01

al2019_fourway_resnet18_600_300:
  extends: al2019_STSAND/(STSAND+STSILT)_resnet18_600
  keys: [SAND, SILT, CLAY, OM]
  maxtime: 3600

al2019_threeway_1920_800_resnet18:
  extends: al2019_threeway_resnet18_600_300
  numpytransforms:
    - "RandomCrop:800,800"
    - HorizontalFlip
    - RandomRotate90
  batchsize: 10
  maxtime: 10800

al2019_threeway_1920_800_resnet18_long:
  extends: al2019_threeway_1920_800_resnet18
  splits: 2
  maxtime: 36000

al2019_fourway_1920_800_resnet18:
  extends: al2019_threeway_1920_800_resnet18
  keys: [SAND, SILT, CLAY, OM]

al2019_fourway_1920_800_resnet18_long:
  extends: al2019_threeway_1920_800_resnet18_long
  keys: [SAND, SILT, CLAY, OM]

al2019_fft1_threeway:
  batchsize: "/8"
  extends: default
  keys: [STSAND, STSILT, STCLAY]
  learningrate: 0.01
  maxtime: 1200
  netclass: FFT1
  normalize: false
  numpytransforms:
    - "SmallestMaxSize:960"
    - "RandomCrop:800,800"
    - HorizontalFlip
    - RandomRotate90
  style: regress

al2019_960_800_fft1_mean_linear1_long:
  extends: al2019_960_800_fft1_mean_linear1
  maxepochs: null
  maxtime: 4800

al2019_threeway_fft_300_200_resnet:
  extends: field26_fft_300_200_resnet
  indexpath: data/al2019/index.yml
  keys: [STSAND, STSILT, STCLAY]
  maxtime: 1800
```