Patterns

CellPress
OPEN ACCESS

Tutorial

Ready, Steady, Go AI: A practical tutorial on fundamentals of artificial intelligence and its applications in phenomics image analysis

Farid Nakhle[1] and Antoine L. Harfouche[1,*]
[1]Department for Innovation in Biological, Agro-food and Forest systems, University of Tuscia, Via S. Camillo de Lellis, Viterbo 01100, Italy
*Correspondence: aharfouche@unitus.it
https://doi.org/10.1016/j.patter.2021.100323

THE BIGGER PICTURE Advances in AI technologies have the potential to significantly increase our ability to turn plant phenomics data into valuable insights. However, performing such analyses requires specialized programming skills commonly reserved for computer scientists. We created an interactive tutorial with free, open-source, and FAIR notebooks that can aid researchers to conduct such analyses without the need for an extensive coding experience. We supplemented it with a practical guide on how to implement AI and X-AI algorithms that augment and complement human experience in classifying tomato leaf diseases and spider mites. Our tutorial is not only applicable to other stresses but also transferable to other plants and research domains, making it possible for researchers from various scientific fields to generate insights into their data. We expect our notebooks to be of high interest to those who want to enhance the performance and sustainability of our agricultural systems through phenomics.

Production Data science output is validated, understood, and regularly used for multiple domains/platforms

SUMMARY

High-throughput image-based technologies are now widely used in the rapidly developing field of digital phenomics and are generating ever-increasing amounts and diversity of data. Artificial intelligence (AI) is becoming a game changer in turning the vast seas of data into valuable predictions and insights. However, this requires specialized programming skills and an in-depth understanding of machine learning, deep learning, and ensemble learning algorithms. Here, we attempt to methodically review the usage of different tools, technologies, and services available to the phenomics data community and show how they can be applied to selected problems in explainable AI-based image analysis. This tutorial provides practical and useful resources for novices and experts to harness the potential of the phenomic data in explainable AI-led breeding programs.

HELLO, WORLD!

In his tutorial memoranda introducing B and C programming languages in 1972 and 1975, respectively, Brian Kernighan wrote an example code that prints "hello, world!" on the computer terminal.[1,2] In 1978, he reused it in "The C Programming Language" book, a worldwide bestseller, which he co-authored with Dennis Ritchie, who originally designed and implemented the language.[3] Ever since, "hello, world!" has become the first program taught when learning a programming language with a purpose to illustrate its basic syntax and verify that the programming environment is properly set up. Seeing these two words on the screen means the code can compile and run correctly.

While this example is used as an illustration for almost every programming language introduced after C, it is also being incorporated in tutorials for older programming languages, such as list processor (LISP), the first high-level functional programming language, designed by John McCarthy in 1958[4]; LISP became predominant over the information processing language (IPL), the first programming language tailored for artificial intelligence (AI) programming, released in the style of low-level assembly language in 1956.[5,6]

Although IPL was introduced in 1956, the roots of AI date back to 1947 when Alan Turing delivered what is perhaps the earliest public lecture on computer intelligence.[7] In 1950, his paper discussed how to build intelligent machines and test their intelligence.[8] Five years later, Allen Newell, Clifford Shaw, and Herbert

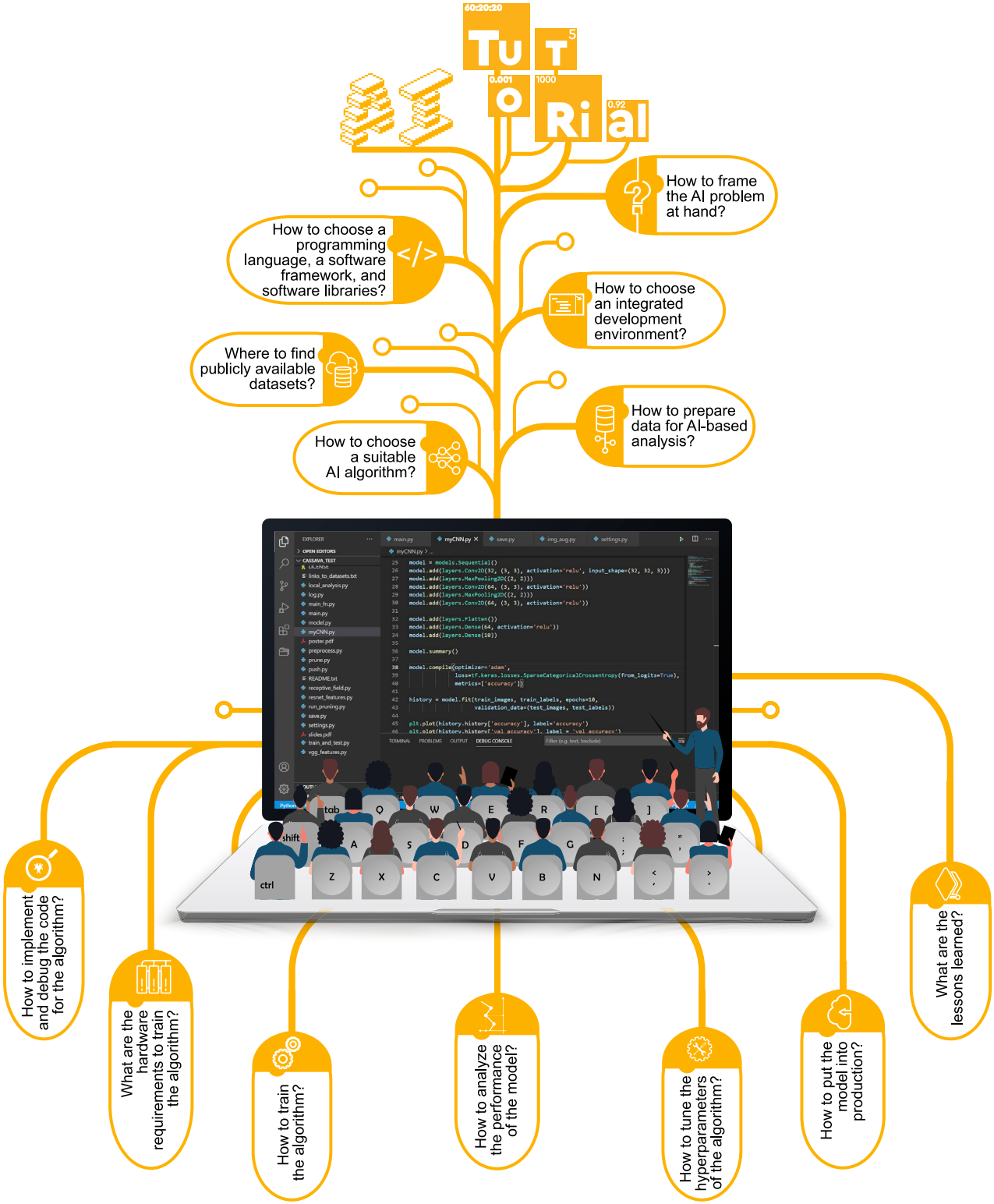

Figure 1. Ready, Steady, Go AI: A practical AI tutorial for the plant data science community

Figure360▷ For a Figure360 author presentation of this figure, see https://doi.org/10.1016/j.patter.2021.100323.

Simon wrote a theorem-proving program, known as the logic theorist, designed to mimic human problem-solving skills.[9] In 1956, their program, often cited as the first AI program,[10] was presented at the Dartmouth summer research project on AI conference hosted by John McCarthy and Marvin Minsky[11] during which McCarthy coined the term AI.

AI continued to flourish until 1973, when the UK science research council published a report, known as the Lighthill report, criticizing AI research for not delivering impactful discoveries.[12] This caused a loss of confidence and henceforth a decline in resources for AI research,[13,14] starting a period known as the AI winter.[15,16] In the 1980s, with Edward Feigenbaum introducing expert systems,[17,18] and the Japanese government funding the fifth-generation computer project,[19] the AI winter ended. The success, however, did not last long as expert systems proved too expensive to maintain and difficult to update.[18] Similarly, the Japanese project failed to meet its goals,[19] marking the second AI winter in the 1990s.[18] Despite this, many researchers continued working with AI under other discipline-specific names, including cognitive systems and intelligent systems.[18] AI continued to advance with the rapid development of computer architectures. Notably, in 1997, International Business Machines (IBM)'s Deep Blue, a chess-playing computer program, defeated world chess champion and grandmaster Gary Kasparov for the first time.[20] In 2012, the AI-based image analysis research marked a significant milestone when Alex Krizhevsky and colleagues published a highly influential paper introducing AlexNet, an AI algorithm that dramatically reduced image analysis error rates and won the ImageNet Large Scale Visual Recognition Challenge with 10.8% fewer errors than that of the runner-up.[21] Next, in 2017, Google's AlphaGo was able to beat Chinese Go champion, Ke Jie.[22] The program simulated millions of Go games by playing against itself and progressively learned the game, accumulating thousands of years of human knowledge during a period of just a few days.[23]

Deep Blue and AlphaGo successfully illustrated how data can contribute to the improvement of AI. As data and AI complement each other, AI-based predictions become better the more quality data the AI is given, but this makes AI programs computationally heavy. In fact, some datasets are too large to fit in computers' memories. Michael Cox and David Ellsworth first coined the term "big data" in 1997 to describe such a scenario.[24] Afterward, big data gets more complex and challenging due not only to its volume but also to field-specific attributes. In this context, Harfouche et al.[25] expanded the dimensions of omics big data to eight Vs: volume, velocity, variety, variability, visibility, value, veracity, and vexing. Nowadays, graphical processing units (GPUs) are accelerating AI with their ability to process multiple tasks simultaneously due to their large number of processing units and high memory bandwidth. With supercomputers breaking the exascale barrier (performing a quintillion floating-point operations per second)[26,27] and the ability of AI to analyze massive amounts of data, researchers can take advantage of big data, rather than relying on smaller representative samples, to extract knowledge and insights without loss of information.

To this end, this tutorial aims to introduce the basic principles for implementing AI and explainable AI (X-AI) algorithms in image-based data analysis (Figure 1) using the PlantVillage dataset as a case study to accurately identify and classify tomato leaf diseases and spider mites, as well as to explain which visual symptoms are used to make predictions. Tomato is the most economically important vegetable crop grown all over the world, but it is highly affected by devastating diseases that hinder its production. This interactive online tutorial provides students and early career researchers who seek an understanding of this rapidly evolving field and want to effectively learn to extract biologically meaningful information from imaging data and the underlying explanation with an easy-to-follow four-step workflow by using existing AI algorithms (Figure 2). It is linked to multiple AI resources that are relevant for experts who are interested in applying AI approaches in phenomics and wish to make use of our open computational notebooks to design and run their own projects. Topics covered include (1) tools, technologies, and services available to the plant data science community; (2) how to automate leaf detection, cropping, and segmentation; (3) how to balance a dataset by means of oversampling and undersampling; (4) how to classify leaf images as healthy, diseased, or spider mites damaged; (5) how to validate the model; and (6) how to generate explanations. As with any new methodology, it is advisable to seek expert assistance and advice before diving too deep.

GETTING READY

Phenomics, the comprehensive large-scale study of high-dimensional phenotypes, is essential to obtaining detailed data of each major aspect of the phenotype and to better understanding plant biology and improve crops.[28–30] However, learning how to use such data more efficiently, easily, and on a large scale is a key challenge to overcome if the implementation of phenomics to improve on-farm response to climate change is to become a reality.[25,26]

Digital phenomics is at the forefront of this tour-de-force effort. Here, we define it as the use of phenome data and metadata to guide decisions along the entire data analytics cycle. More broadly, it cycles between data collection and/or selection, preprocessing, analysis, and interpretation to produce new knowledge and, consequently, societal benefits. Importantly, as imaging is a promising source of phenotypic data, digital phenomics is not restricted to image analysis but can span the environment-to-phenotype map, which allowed the exploration of a wide range of organisms' phenotypic responses to environmental conditions,[31] as well as the genotype-to-phenotype map, which describes how genetic variation affects phenotypes.[32–34] These aspects of digital phenomics are crucial to accelerate breeding.

Two of the main priorities for digital phenomics are (1) to develop the tools and resources that facilitate the data analytics cycle, and (2) to deploy digital platform technologies that provide powerful solutions not just to researchers but, more importantly, to breeders and farmers. The need for a bigger-science approach is most apparent in the development of digital phenomics approaches in which digitalization of phenotyping enables, improves, and transforms research design, innovation models, and entrepreneurship activities, by leveraging digital technologies and data.

Data analytics is crucial for working with large-scale phenomics data. Traditional statistical methods are limited in analyzing high-dimensional data and are unable to capture complex

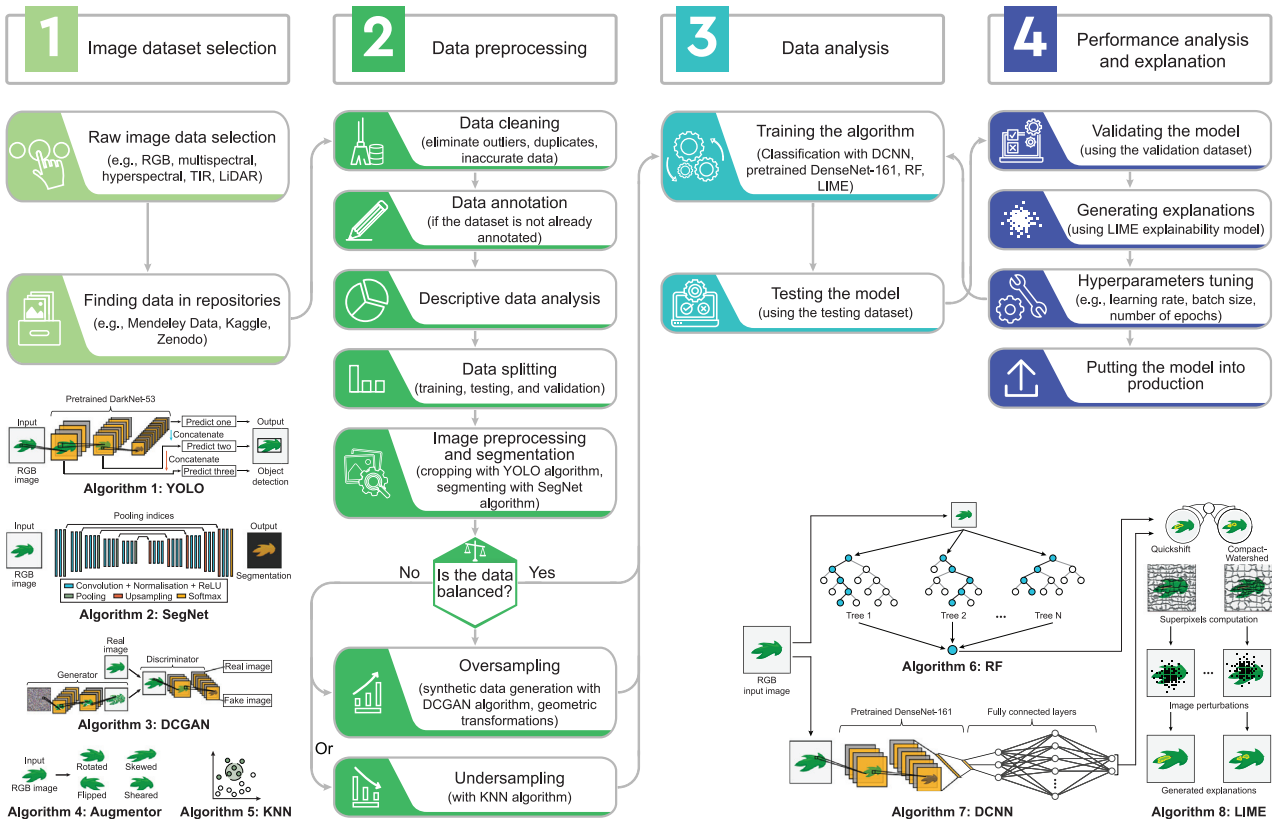

Figure 2. Schematic overview of the proposed X-AI-based image analysis workflow for a typical digital phenomics experiment from selection through preprocessing, analysis, and explanation

relationships between the predictor (independent) and response (dependent) variables.[35] The advancement of computing technologies and power as well as AI-driven data analytics offer the possibility that digitalization of phenomics can provide new solutions to these complex challenges. Crucially, AI, in its various techniques (e.g., machine learning [ML], deep learning [DL], and ensemble learning [EL]), could be an alternative approach. For example, AI has shown impressive results in plant stress phenotyping, as reviewed by Singh and coworkers,[36,37] and in predicting crop yield[38,39] and plant phenotypes.[40–42] However, complex AI models typically operate as black boxes and require a leap of faith to believe their predictions.[25,43] Fortunately, efforts to develop approaches that make their inner workings understandable to humans have improved our ability to explain their predictions[43] and introduced X-AI. With X-AI, models operating as black boxes become transparent, establishing trust between humans and AI-based predictions. While a wide range of X-AI approaches and algorithms have been developed and covered by Azodi et al.,[43] Rudin et al.,[44] and Barredo Arrieta et al.,[45] this tutorial focuses on post hoc model-agnostic algorithms such as local interpretable model-agnostic explanations (LIME), which is the X-AI algorithm employed in this study. Such algorithms are used to explain any black box models by extracting the relationships between their input values and predictions to approximate their predictive behavior. Post hoc algorithms are commonly categorized by the scope of their explanations, which can be global or local. While the former describe the overall extracted relationships based on the entire model behavior, the latter, which LIME provides, reveal the rationale behind a specific prediction.[43,45]

Some basics

ML

ML traces back to 1943, when Warren McCulloch and Walter Pitts modeled an electrical circuit to describe how neurons in the brain might work.[46] Their research inspired the design of the artificial neural network (ANN) algorithm, in which layers of connected nodes called artificial neurons mimic how human brains process information.

ML is a multidisciplinary approach to data analysis that makes use of probability theory, statistics, decision theory, and mathematical optimization[47] to learn from data in order to extract important information, find hidden patterns, and make associations and predictions.[25] It can be classified into either unsupervised, supervised, or semi-supervised learning. Unsupervised learning algorithms find patterns and learn meaningful relationships in unlabeled data to describe its underlying structure. They can be categorized into clustering, association, and dimensionality reduction. On the other hand, supervised learning algorithms learn from labeled data to assign a label for previously unseen data. Supervised learning can be further divided into classification, when predicting a class label, and regression, when predicting a numerical value. For example, if we were to write a classification program to distinguish

between healthy and diseased tomato leaves, all we have to do is to use a large number of leaf images, each labeled as healthy or diseased, as input for a classification algorithm. From labeled images, the algorithm being trained accumulates experience and creates its own set of rules to ultimately predict the correct class of an unlabeled image. Semi-supervised learning falls in between the two, where only part of the data needs to be labeled.

Both Turing and McCarthy suggested that the combination of human knowledge with ML is central to the development of AI research.[8,48] ML algorithms can integrate knowledge in the form of equations and logic rules to enhance learning performance.[49] Consequently, methods such as logic-based ML make use of inductive logic programming[50] to integrate information and patterns extracted from data with human knowledge in order to make predictions.[51,52]

Various ML algorithms have been developed and used in plant phenotyping, as reviewed by Singh et al.[47] and van Dijk et al.[53] While unsupervised, supervised, and semi-supervised learning can be useful depending on the goal to be achieved, in this tutorial, we focus on supervised learning, with an aim to provide researchers with ways to identify and classify healthy, diseased, and spider mite-damaged tomato leaf images.

DL

ANN has undergone significant improvements, mainly by increasing its number of layers.[54] This multilayered architecture is known as deep neural networks (DNNs). DL is a class of ML that extracts features from annotated data and trains algorithms via DNNs to provide predictions based on discovered patterns.[25,55]

Historically, the earliest efforts in developing DL algorithms came from Alexey Ivakhnenko and Valentin Lapa in 1965, when they created the group method of data handling (GMDH) algorithm which gradually increments the number of layers of an ANN based on evaluation of its performance.[56] In 1979, Kunihiko Fukushima designed an ANN with multiple data downsampling (pooling) and filtering (convolutional) layers called Neocognitron. It was capable of analyzing images[57,58] and led to the birth of deep convolutional neural networks (DCNNs). Slow progress was being made in this field until 2012 when AlexNet, trained to classify 1.2 million high-resolution images, achieved a top-five error rate of 15.3%, making a major jump in the accuracy of image analysis.[21] Since then, DCNNs became the default approach for most AI-based image analysis tasks.[59] Various DL algorithms have been developed and used in plant phenotyping.[36,37]

EL

EL is a technique where a set of algorithms are individually trained to solve the same problem, having their output combined to improve predictive performance and reduce error variance.[25,60]

Tracing the origin of EL proved difficult as the idea of training multiple algorithms and combining their predictions has been used for a long time.[61] However, EL research was highly influenced by two main events in 1990.[61] The first was when Robert Schapire introduced boosting by theoretically proving that the prediction accuracy of learners that are not highly accurate, known as weak learners, can be boosted by their combination.[62] The second was when Lars Hansen and Peter Salamon conducted applied research and found that combining the predictions of a set of algorithms is often more accurate than using a single one.[63] Later, in 1996, Leo Breiman introduced bagging, which stands for bootstrap aggregating, where the algorithms of an ensemble are trained on different randomly sampled subsets of data.[64]

The combination of algorithms to create an ensemble can be homogeneous, by applying the same algorithm for all combined estimators (e.g., decision trees), or heterogeneous, by combining a variety of algorithms (e.g., support vector machines with logistic regression). Notably, the combination of multiple DL algorithms is sometimes referred to as deep EL.

The remarkable flexibility and adaptability of EL helped the proliferation of its application in plant phenotyping where it has been used for various tasks, including biotic stress identification and classification,[65,66] aboveground biomass estimation,[67,68] and performance prediction of crosses in breeding.[69,70]

GETTING SET

At the end of his speech, on February 9, 1941, Winston Churchill said "give us the tools, and we will finish the job" in an attempt to raise the morale of the British public during World War II. While the task of implementing AI algorithms is not to be compared with war, choosing the right tools, technologies, and services is paramount to its success. However, the number of available programming languages, software frameworks, software libraries, and integrated development environments (IDEs) can be daunting. Here, we guide researchers through the options of making a suitable choice.

Programming languages and IDEs

Programming languages are tools used to write instructions for computers to follow. They fall into two categories: low level and high level. Low-level languages are close to binary; they have the advantage of being faster to execute and use less computer memory. In contrast, high-level programming languages are closer to how humans communicate; this makes them easier to use compared with low-level languages. However, they are slower to run as, prior to execution, they get translated into machine code through a compiler that scans the entire program and translates it all at once, or an interpreter, which translates just one statement at a time. However, as computing power continues to increase, the runtime difference between low-level and high-level languages becomes negligible. Examples of high-level programming languages include Python, C, and R.

A programming language is said to be Turing complete if it is possible to implement any algorithm with it, regardless of its performance.[71] In this sense, any Turing complete programming language can be used to implement any AI algorithm. Unfortunately, a single ideal programming language does not exist. Choosing one requires consideration of all of its aspects, benefits, and shortcomings in respect to the task at hand. It is advisable to consider the ecosystem around it, such as its community, contributors, and available software frameworks and libraries (Table 1 and Table 2). It is also prudent to choose the language that is adopted in the user's professional environment, which increases code reusability. Well-established, stable languages are

Table 1. Representative list of available open-source software frameworks and programming languages for the implementation of AI algorithms

Software framework	Programming languages	Operating systems	GPU computing support[a]	Distributed training support	Availability of pretrained models[b]	Software license	Website
Deep ML frameworks[c]							
Tensorflow & Keras[d]	Python, JavaScript, C++, Java, Go, Swift (early release)	Linux, Windows, macOS	Linux, Windows	multiple GPUs, multiple machines	yes	Apache 2.0	www.tensorflow.org
PyTorch	Python, C++, Java	Linux, Windows, macOS	Linux, Windows	multiple GPUs, multiple machines	yes	BSD-3	pytorch.org
Apache MXNet	Python, C++, Scala, Julia, Clojure, Java, R, Perl	Linux, Windows, macOS	Linux, Windows	multiple GPUs, multiple machines	yes	Apache 2.0	mxnet.apache.org
H_2O	Python, Scala, Java, R	Linux, Windows	Linux, Windows	multiple GPUs, multiple machines	no	Apache 2.0	www.h2o.ai
Deeplearning4j	Java, Groovy, Scala, Kotlin, Clojure	Linux, Windows, macOS, Android	Linux, Windows	multiple GPUs, multiple machines	yes	Apache 2.0	deeplearning4j.org
Chainer	Python	Linux	Linux	multiple GPUs, multiple machines	yes	MIT	chainer.org
PaddlePaddle	Python	Linux, Windows, macOS	Linux, Windows	multiple GPUs, multiple machines	yes	Apache 2.0	www.paddlepaddle.org
SINGA	Python, C++	Linux, Windows, macOS	Linux, Windows	multiple GPUs, multiple machines	no[e]	Apache 2.0	singa.apache.org
Flux	Julia	Linux	Linux	multiple GPUs	no	MIT	fluxml.ai
OpenNN	C++	Linux, Windows, macOS	Linux, Windows	no	no	LGPLv3	www.opennn.net
Dlib	Python, C++	Linux, Windows, macOS	Linux, Windows	multiple GPUs	no	BSL-1.0	dlib.net
MLBox	Python	Linux, Windows, macOS	no	no	no	BSD-3	github.com/AxeldeRomblay/MLBox

(*Continued on next page*)

Table 1. ***Continued***

Software framework	Programming languages	Operating systems	GPU computing support[a]	Distributed training support	Availability of pretrained models[b]	Software license	Website
ML frameworks							
scikit-learn	Python	Linux, Windows, macOS	no	no	no	BSD-3	scikit-learn.org
Apache Mahout	Java, Scala	Linux, Windows, macOS	Linux, Windows	multiple GPUs, multiple machines	no	Apache 2.0	mahout.apache.org
xLearn	Python, R	Linux, Windows, macOS	no	multiple machines	no	Apache 2.0	github.com/aksnzhy/xlearn
Shogun	Python, Octave, R, Java, Scala, Lua, C#, Ruby	Linux, Windows, macOS	no	no	no	GPLv3	www.shogun-toolbox.org
SystemDS	Python, Java	Linux, Windows, macOS	Linux, Windows	multiple GPUs, multiple machines	no	Apache 2.0	systemds.apache.org

BSD, Berkeley Software Distribution; BSL, Boost Software License; GPL, GNU General Public License; LGPL, GNU Lesser General Public License; MIT, Massachusetts Institute of Technology.

[a]While all software frameworks support CPU-based analysis, some support GPU-based analysis, which can make key computations very fast, particularly convolution operations used in image analysis.

[b]The availability of pretrained models allows giving an AI model a warm start by applying information learned from another previously trained model using a process known as transfer learning. For example, frameworks such as Tensorflow with Keras, PyTorch, and PaddlePaddle allow the transfer of a pretrained DenseNet-161 model to a DCNN model, and DarkNet-53 pretrained model to a YOLO model (Figure 2).

[c]Deep ML frameworks can support both ML and DL AI algorithms.

[d]Keras is the high-level application programming interface (API) of TensorFlow, allowing users to train, evaluate, and use AI models in just a few lines of code.

[e]While SINGA does not provide pretrained AI models, it supports transferring models in Open Neural Network Exchange (ONNX) format, an open-representation format for AI models that enables users to use them across different software frameworks.

Table 2. Representative list of available open-source software libraries and IDEs to aid the implementation of AI algorithms

Software library/IDE	Programming languages	Description	Website
Software libraries			
OpenCV	Python, R, C, C++, Java, Julia, Closure	helps applying various image processing operations such as image resizing, object detection, and geometrical transformations	opencv.org
Plotly	Python, R, Julia	enables plotting charts such as histograms, bar charts, line charts, and boxplots	plotly.com
NumPy[a]	Python	provides multidimensional arrays and various mathematical functions and statistical operations to operate on these arrays	numpy.org
pandas	Python	offers data structures and operations that enable the analysis and manipulation of different kinds of data, such as tabular, time series, and arbitrary matrix data	pandas.pydata.org
Matplotlib[b]	Python	helps generating different types of charts such as line charts, scatterplots, heat maps, and bar charts	matplotlib.org
Seaborn	Python	based on Matplotlib, it enables data visualizations by providing a simpler API	seaborn.pydata.org
Bokeh	Python	allows creating web-based data visualizations and charts ranging from simple plots to complex data dashboards	bokeh.org
split-folders[c]	Python	allows splitting a dataset into training, validation, and test sets	github.com/jfilter/split-folders
Augmentor[d]	Python	oversamples an image dataset by applying transformations to image geometries	augmentor.readthedocs.io/en/master
DataExplorer	R	automates the descriptive data analysis process by generating basic statistics for a dataset (e.g., types of variables, missing values) and plotting distributions of variables and their correlation matrices	cran.r-project.org/web/packages/DataExplorer
ggplot2	R	helps plotting charts such as line charts, histograms, scatterplots, density plots, boxplots, and bar charts	ggplot2.tidyverse.org
tidyr	R	provides table data structures and organizes variables in columns and values in rows	tidyr.tidyverse.org
magick	R	helps editing and converting image data from and to a variety of formats, including PNG, JPEG, and TIFF. Enables resizing and applying geometric transformation to images	cran.r-project.org/web/packages/magick

(*Continued on next page*)

Table 2. ***Continued***

Software library/IDE	Programming languages	Description	Website
Magick++	C++	a C++ API for the magick library	imagemagick.org/Magick++
accuracy-evaluation-cpp	C++	helps calculating performance metrics for classification algorithms, including accuracy, error rate, precision, recall, and F score	github.com/ashokpant/accuracy-evaluation-cpp
matplotlib-cpp	C++	a C++ API for the Matplotlib library	github.com/lava/matplotlib-cpp
Armadillo	C++	provides data structures and linear algebra operations for vectors, matrices, and cubes	arma.sourceforge.net
ND4J	Java	provides multidimensional arrays and various linear algebra functions to operate on them	github.com/deeplearning4j/nd4j
matplotlib4j	Java	a Java API for the Matplotlib library	github.com/sh0nk/matplotlib4j
cv4j	Java, Android	allows the application of various image filtering techniques, including the median filter for image denoising, sharp filter for image sharpening, and Laplacian filter for edge detection	github.com/imageprocessor/cv4j
Tables.jl	Julia	offers data structures and operations that enable the analysis and manipulation of tabular data	github.com/JuliaData/Tables.jl
Plots	Julia	brings plotting functionalities to Julia from several plotting libraries, including Pyplot from the Python Matplotlib library	docs.juliaplots.org/latest
Augmentor.jl	Julia	a Julia API for the Augmentor library	github.com/Evizero/Augmentor.jl
Vegas	Scala	helps generating different types of charts, such as line charts, scatterplots, heat maps, and bar charts	github.com/vegas-viz/Vegas
Breeze	Scala	provides data structures for vectors and matrices, and statistical functions to calculate descriptive statistics, such as mean, variance, and standard deviation	www.scalanlp.org
kixi.stats	Closure	generates descriptive statistics such as mean, variance, and standard deviation, and provides functions for performing statistical tests, such as t test, z test, and chi-square test	cljdoc.xyz/d/kixi/stats
Neanderthal	Closure	provides data structures for vectors and matrices in addition to a variety of functions that support their manipulation	neanderthal.uncomplicate.org/
Hanami	Closure	helps plotting charts such as bar charts, contour plots, and tree layouts using the declarative languages Vega and Vega-Lite	github.com/jsa-aerial/hanami

(*Continued on next page*)

Table 2. *Continued*

Software library/IDE	Programming languages	Description	Website
panthera	Closure	an API that imports NumPy and pandas functionalities from Python to Closure	github.com/alanmarazzi/panthera
IDEs			
Visual Studio Code	Python, R, C, C++, C#, Julia, Java	is a cross-platform code editor[8] offering code syntax highlighting, smart completion, and debugging. It helps reviewing differences in code versions and integrates with SCM services. It is extensible, providing the possibility to add new languages, themes, and debuggers	code.visualstudio.com
Eclipse	Python, R, C, C++, Java, C#, Clojure, Julia, Scala	is composed of plugins and designed to be extensible using additional plugins. It is customizable, cross-platform, and offers code syntax highlighting, code debugging, and the ability to review differences in code versions	www.eclipse.org/ide
NetBeans	C, C++, Java	provides code syntax highlighting, templates, and a range of tools for code debugging and refactoring, and integration with SCM services. It is cross-platform and is able to identify and fix common problems in Java code	netbeans.org
PyCharm Community Edition	Python	provides smart code completion, error highlighting, and code debugging capabilities. It is cross-platform and customizable, providing various code editor color templates	www.jetbrains.com/pycharm/
IDLE	Python	offers an interactive cross-platform Python interpreter with code syntax highlighting, smart indentations, and code debugging	docs.python.org/3/library/idle.html
IntelliJ IDEA Community Edition	Java, Scala, Android	provides code syntax highlighting, debugging, tools for code refactoring, and a unified interface for integration with SCM services. It is customizable, cross-platform, and allows reviewing differences in code versions	www.jetbrains.com/idea
Code::Blocks	C, C++, Fortran	is composed of plugins and designed to be extensible using additional plugins. It is cross-platform and provides code syntax highlighting, smart indentations, and code debugging	www.codeblocks.org

(*Continued on next page*)

Table 2. *Continued*

Software library/IDE	Programming languages	Description	Website
KDevelop	Python, C, C++	provides code syntax highlighting and code completion. It is customizable, cross-platform, and allows code debugging. It supports integration with various SCM services	www.kdevelop.org
RStudio	R	offers code syntax highlighting and supports direct code execution. It is cross-platform and provides tools for debugging and viewing code execution history	www.rstudio.com

PNG, portable graphics format; TIFF, tag image file format.
[a]By loading image data as NumPy arrays, various image processing tasks can be performed using functions provided by other software frameworks and libraries; in this tutorial, NumPy aided the implementation of the DCNN, YOLO, SegNet, and DCGAN algorithms.
[b]Aided the visualization of the confusion matrix used to evaluate the performance of the DCNN algorithm.
[c]Helped the random splitting of PlantVillage dataset, used in this tutorial, into training, testing, and validation sets.
[d]Assisted in oversampling PlantVillage to ensure the balance between different classes and avoid overfitting.
[e]While it is a code editor, its support for development operations, including debugging and version control, and its high extensibility and customizability level allow it to be configured as an IDE.

often a convenient choice. For example, Julia combines the interactivity of scripting languages, such as Python and R, with the speed of compiled languages such as C.[72] It was originally designed for high-performance scientific computing and data analysis.[73] However, being in early-stage development, Julia has a limited amount of software frameworks and libraries.[73] Its influence is far less than that of other popular programming languages such as Python and R, which are considered the standard tools of the trade in research and industry, with Python being a more popular choice among biologists[74]; they both have an easy-to-learn syntax and have no major speed differences when it comes to typical data operations.[74]

The choice of a programming language is often complemented by the choice of an IDE. IDEs allow researchers to write codes quickly and increase their productivity by combining code editing, compiling, and debugging into a single application. The choice of IDEs is personal. The reason to pick one over the others depends on the tasks required to be performed. Common features to look for are code autocompletion, syntax highlighting, code refactoring, and more that could increase productivity (Table 2). For example, while PyCharm provides smart code completion, error highlighting, and code debugging capabilities for Python, Visual Studio Code supports various programming languages and can help review differences in code versions and integrate with source code management (SCM) services.

Software frameworks

Software frameworks are a working environment that helps researchers quickly implement or design algorithms without compromising quality. They provide generic codes that are tested and optimized for others to reuse, extend, and customize to their needs. Additionally, they include functions that can be used to process input, manage devices, and interact with system software. For example, using a software framework can remove the burden of writing a code that loads images from a disk drive. Instead, a pre-written code provided by the framework can be used. Various software frameworks are available to help the implementation of AI algorithms (Table 1).

Software libraries

Similar to software frameworks, software libraries consist of pre-written codes. They can be used to import a specific functionality without writing a code for it. For example, when implementing an AI algorithm with Python, importing NumPy provides multidimensional arrays and various mathematical functions to operate on them, eliminating the need to code such complex functionality.

Unlike software frameworks, libraries typically focus on a narrow scope where they perform a set of specific, well-defined operations. Technically, the key difference between them is known as the inversion of control (IoC), meaning that, with software libraries, researchers are in control of the code as they use library functions whenever they see fit. However, software frameworks require them to follow specific workflow and software design patterns. Table 2 provides a list of software libraries that can come in handy for implementing AI algorithms.

GOING AI

AI can potentially revolutionize many aspects of image analysis in phenomics, but it would have a greater impact when everyone can access it, including not only computer scientists but also researchers across disciplines. While the emergence of freely available and open-source AI resources is opening the way toward wider AI accessibility, the substantial expertise required for its adoption remains a hurdle. Adopters do not need to have extensive experience in programming, but, in addition to their domain knowledge, they should be open to learning a programming language and writing some codes for their experiments. Going AI is set to avoid exposing tutees to heavy jargon and tedious definitions, but, instead, to tutor them on the concepts behind AI algorithms relevant to the task at hand and equip them with the right skills that make them comfortable using those resources. This would enable them to gain the practical know-how needed to address new problems in phenomics image analysis. When they complete this tutorial experience, they should be in a position to ask biological questions, analyze available resources, and approach several types of plant image datasets using various AI algorithms. Although a single workflow cannot be considered the best choice as different algorithms are to be assessed for different tasks and objectives, our approach is in agreement with common image-based analytical methods, along the lines of those described elsewhere,[36,37,75] as essential key steps in the workflow of image analysis. The four steps of this workflow are illustrated in Figure 2. For each of the eight algorithms used, we created an interactive notebook runnable on Google Colab, or any other platform supporting Jupyter notebooks, and made it available on our public GitHub repository (see section "data and code availability"). The repository includes the notebooks and corresponding ReadMe files that instruct researchers through the process of running our code either to reproduce the results or to run the implemented algorithms with a new dataset. A continuous integration (CI) framework using the Travis CI service (https://travis-ci.org) was added so that unit tests, implemented using pytest (https://pytest.org), can run automatically when code updates are introduced. This provides a safeguard against updates that can break existing functionality by generating a report showing which tests have passed or failed. The coverage of the test is measured using the Coverage.py software library (https://coverage.readthedocs.io/en/coverage-5.5), which reports the amount (percentage) of code covered by existing unit tests. In addition, we have created a set of self-test quizzes and hands-on practices and exercises to provide researchers with opportunities to augment their learning by testing the knowledge they have acquired and by practically applying the concepts explained. For the quizzes, two Google Forms of multiple-choice questions (MCQs) were developed. The first asks questions about the basics covered in this paper, while the second, intended for the more experienced researchers, includes more advanced questions and is based on real-life scenarios that we have faced while developing this tutorial. This will help researchers identify good practices and develop troubleshooting skills that can be useful when performing AI-based image analysis.

For the practices and exercises, three computational notebooks were developed to get researchers involved in coding. The first, intended for beginners, asks users to complete simple parts of a provided code in order to run the tasks included in our workflow; the second, intended for intermediates, involves users in writing more codes for the same tasks and modifying parts of it to help them practice; the third notebook is intended for the more experienced users where they are asked to write codes that reuse previously learned models to preprocess images of apple leaves included in the PlantVillage dataset and containing four classes of diseases (apple scab, black rot, cedar apple rust, and healthy). Next, the notebook requires them to train a new model that is able to identify and classify apple diseases and explain its decisions. All three notebooks contain the solutions stored in hidden code cells after each task and are available on our public repository along with links to the MCQs.

Step 1: Image dataset selection

Initially, it is important to frame the biological question at hand. This helps researchers identify what task their AI algorithm should perform and guide the choice of the input data. For example, red-green-blue (RGB) imaging data can be used to monitor plant stress,[76] while light detection and ranging (LiDAR) data allow measurement of plant architecture, canopy height, and growth rates.[25,77]

Publicly available plant phenotyping datasets have a 2-fold advantage: on the one hand, the phenomics data science community is given access to valuable data whose collection can be both time consuming and expensive, thus providing the foundation for practical AI-based image analysis applications; on the other hand, freely available benchmarking datasets can serve as a basis to compare the performance of AI algorithms under reproducible settings.

In this tutorial, we performed our analysis using the publicly available PlantVillage tomato dataset (Pennsylvania State University, United States).[78] The dataset includes a total of n_{tomato} = 18,160 individually labeled and non-overlapped RGB images divided into 10 classes (Figure 3): class 0 (healthy, n_h = 1,591), class 1 (bacterial spot, n_{bs} = 2,127), class 2 (leaf mold, n_{lm} = 952), class 3 (*Septoria* leaf spot, n_{sls} = 1,771), class 4 (target spot, n_{ts} = 1,404), class 5 (early blight, n_{eb} = 1,000), class 6 (late blight, n_{lb} = 1,909), class 7 (mosaic virus, n_{mv} = 373), class 8 (yellow leaf curl virus, n_{ylcv} = 5,357), and class 9 (spider mites, n_{sm} = 1,676). All images were in joint photographic expert group (JPEG) format and had a standard size of 256 × 256 pixels.

Step 2: Data preprocessing

Data preprocessing aims to eliminate noisy (e.g., blurred images), incomplete (e.g., unannotated images), and duplicate data from a dataset to help AI algorithms learn relevant features that ultimately enhance their accuracy. This step starts with data cleaning, which is the process of inspecting data to remove irrelevant (e.g., images that do not contain a leaf) or wrongly annotated (e.g., diseased images labeled as healthy) images. In case images were not annotated, adding corresponding annotations to images would then be necessary. Both processes are often done manually, preferably with the help of domain experts (e.g., a plant pathologist). Indeed, the tomato dataset has been cleaned and provided with image-level annotations (i.e., assigning a class to each image) by plant pathology experts.[78] Next, a descriptive data analysis must be performed to have a better understanding of data

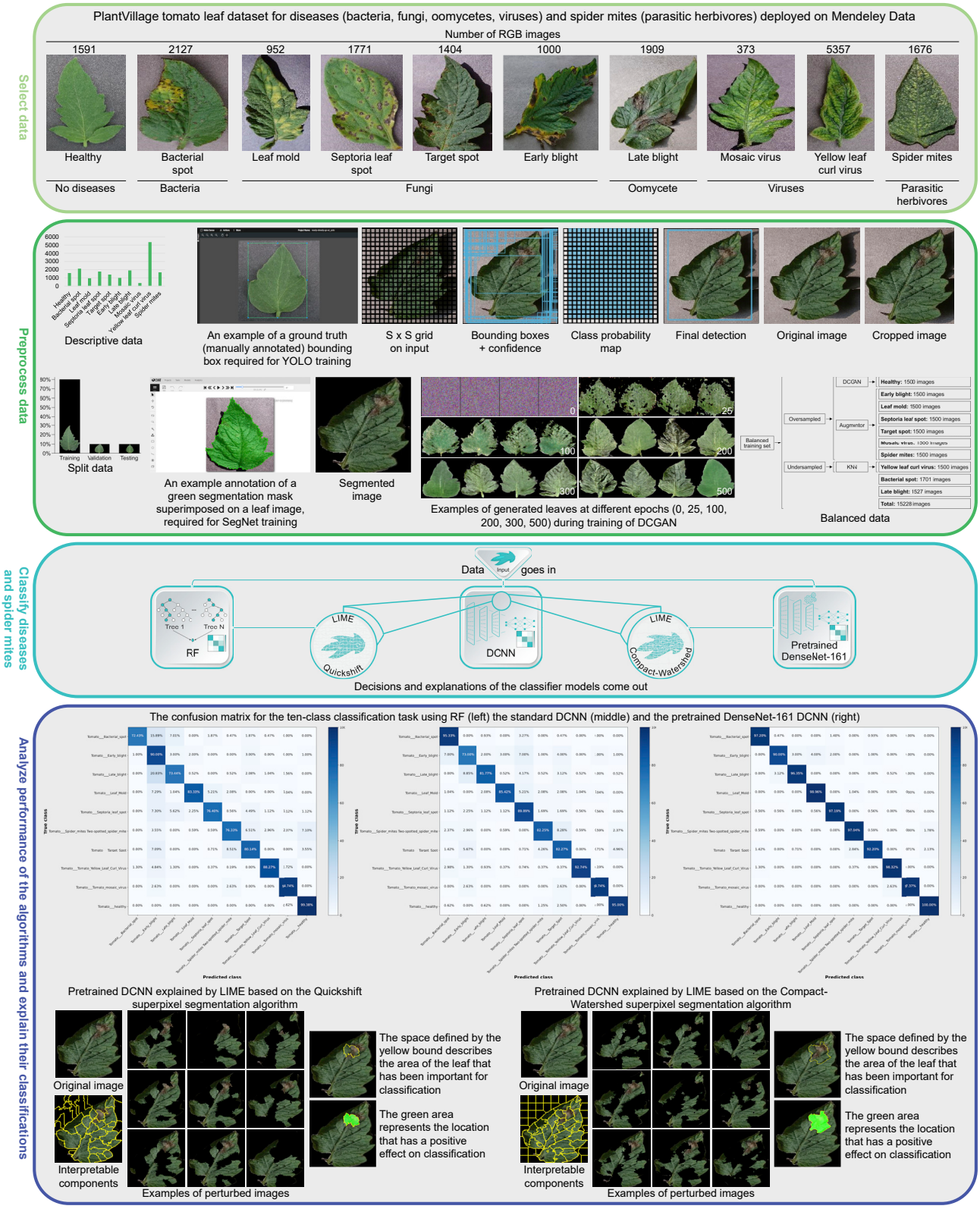

Figure 3. Visualization of the tutorial outputs at every step of the proposed X-AI-based image analysis workflow

characteristics, such as its frequency distribution (i.e., the number of images in each class). A simple Python code was developed to iterate over the directories of the 10 classes, counting the number of images in each. This analysis showed that classes of tomato images are imbalanced (Figure 3). This calls for data balancing, done by either oversampling the minority or undersampling the majority classes. However, prior to balancing, the dataset must be split into training, validation, and testing sets to prevent overlap of oversampled images across them. While the training set is used to fit the model, the validation set evaluates it during hyperparameters tuning, and the test set serves to provide an unbiased evaluation of the final model. Here, we randomly allocated 80% of images for training (n_{train} = 14,523), 10% for validation (n_{val} = 1,812), and 10% for testing (n_{test} = 1,825), using the split-folders Python software library. Then, we applied both oversampling and undersampling to balance the dataset. We used two techniques for oversampling: first, by creating random transformations to image geometries using the Augmentor Python software library, and second, by synthesizing leaf images using a deep convolutional generative adversarial network (DCGAN) as suggested by Perez and Wang,[79] concluding that the likely best strategy for oversampling would be to combine the geometrically transformed data with synthesized data. Undersampling was performed using the k-nearest neighbors (KNN) algorithm.

It is worth noting that datasets in public repositories may include images that contain more than one object, typically belonging to background objects (e.g., stem, overlapping leaves) that are not of interest for an AI algorithm to learn. This could negatively affect its training, but it can be solved by image cropping and/or segmentation. However, this process can take a considerable amount of time and effort. Many types of plant image processing software, such as plant computer vision (PlantCV),[80,81] Image Harvest, and HTPheno,[82] can help with these tasks and extract features from images by automating complex techniques such as histogram thresholding and color analysis.[83] In this regard, Lobet et al.[84] developed an online database (http://plant-image-analysis.org) indexing both commercial and open-source plant image processing software. While such software makes it easy for researchers to get started, more advanced use cases still require them to develop their own. Here, we showed how cropping and segmentation can be automated using the you only look once (YOLO) and SegNet AI algorithms, respectively.

Finally, before diving into image preprocessing and analysis algorithms, it is fundamental to understand how computers process an RGB image. Computers denote every pixel in an image as a 1 × 3 matrix, representing the three colors R, G, and B with each being an 8-bit integer leading to 256 (2^8) possible states for every color. These states represent the brightness of each color, with 0 being the darkest and 255 being the brightest. For example, a matrix with values R = 255, G = 255, and B = 255 would lead to a white pixel.

Algorithm 1: YOLO

YOLO was first introduced in 2016 to perform object detection and localization in images and videos.[85] In 2017 and 2018, improved versions of the algorithm were released, namely YOLOv2[86] and YOLOv3,[87] respectively. YOLO starts by dividing the input image into an S × S grid, forming S^2 cells (Figure 3). For every object in the image, for example a leaf, the grid cell associated with its center is responsible for detecting it. This is done by having the cell predict a number of rectangles that enclose the object, known as bounding boxes. These predictions have five components: x, y, w, h, and confidence. While x and y are the coordinates representing the center of the bounding box, w and h represent the width and height measures that describe its dimensions. Finally, the confidence score is equal to the intersection over union (IOU) between the predicted and the ground truth (manually annotated) bounding box (Figure 3).

When tested on the Microsoft common objects in context (COCO) large-scale object detection dataset,[88] YOLOv3 was able to detect objects faster and more accurately than other relevant object detection algorithms, including faster region-CNN (Faster R-CNN), single shot multibox detector (SSD), and RetinaNet, scoring the highest mean average precision (mAP) in the shortest time.[87]

We used YOLOv3 to automatically detect, localize, and crop the leaves. We randomly selected 1,000 images from all 10 classes and manually annotated them by drawing bounding boxes around the leaves using the MakeSense annotation tool (https://makesense.ai). These images were then used to train YOLOv3, which uses a DCNN architecture (see algorithm 7) named DarkNet-53, pretrained on the ImageNet dataset. An example architecture of YOLO is illustrated in Figure 2. After the algorithm has completed 300 training epochs (iterated over the annotated images 300 times), the learned model was used to detect and localize leaves for the rest of the dataset. Once a final detection and a class probability map were generated for each image (Figure 3), a custom code was used to crop it, keeping only the leaf.

Algorithm 2: SegNet

SegNet is a DCNN-based algorithm (see algorithm 7) that uses a modified visual geometry group (VGG)-16 architecture to perform pixel-wise image segmentation,[89] which involves assigning a class to every pixel in an image. SegNet is composed of encoders and decoders where the encoder network consists of 13 convolutional layers followed by their corresponding decoding layers forming the decoder network. The encoder-decoder network is finally followed by a classification layer that uses the softmax activation function. While the encoder network performs convolutions with various filters to produce a set of feature maps, the corresponding decoders upsample them and generate a segmentation mask for the image. Finally, the softmax classifier assigns a class to each pixel. The described SegNet architecture is illustrated in Figure 2.

Quantitative comparison of deep networks for semantic pixel-wise image segmentation using the Cambridge-driving labeled video database (CamVid)[90] showed that SegNet learned to perform better in a shorter time than competing algorithms such as DeepLab, fully convolutional network (FCN), and deconvolution network (DeconvNet).[89]

In this tutorial, 150 images, chosen randomly from the 10 classes, were annotated with segmentation masks using the computer vision annotation tool (https://cvat.org) to train SegNet on leaf segmentation (Figure 3). The trained model was then used to segment the remaining images by deleting the background around the leaves (Figure 3).

Algorithm 3: DCGAN

DCGAN is an extension of the generative adversarial network (GAN) that was designed in 2014 to generate new data instances that resemble training data.[91] The idea of GAN was to pair two learning models, typically two ANNs, named generator and discriminator, where the former learns to produce synthetic data and the latter learns to distinguish true data from the output of the generator. During training, the generator tries to deceive the discriminator by synthesizing better data, while the discriminator becomes a better classifier. The equilibrium of this zero-sum game is reached when the discriminator can no longer distinguish real images from synthetic ones.

DCGAN, released in 2016, is very similar to GAN, except that it uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively, making it more suitable for synthesizing imaging data.[92] While there exist other algorithms to synthesize data, such as the synthetic minority oversampling technique (SMOTE)[93] and adaptive synthetic sampling (ADASYN),[94] they are mostly suitable for tabular data, which restricts their application in high-dimensional complex image data.[95] DCGAN instead can generate synthetic images with high visual fidelity. When it was first introduced, Radford et al.[92] identified an architecture that allows training the algorithm to generate higher-resolution images. We adopted this architecture by (1) using stride convolutions instead of pooling layers (see algorithm 7); (2) using batch normalization in both the generator and discriminator; (3) refraining from using hidden layers in the fully connected network in both the generator and discriminator; (4) using the rectified linear unit (ReLU) activation function in the generator for all layers except for the output, which uses the hyperbolic tangent (Tanh) activation function; and (5) using LeakyReLU activation function in the discriminator for all layers. Finally, the binary cross entropy was employed as a loss function.

Figure 2 shows an example architecture of DCGAN. Here, we provided DCGAN with the healthy class images of the training set ($n_{h \subset train}$ = 1,272) to train it on generating synthetic leaves, and, thus, oversampling the set to 1,500 images after 500 training epochs (Figure 3), on a learning rate of 0.0002 as suggested by Radford et al.[92]

Algorithm 4: Augmentor

Another approach to oversampling is using geometric transformations to generate new images. This helps models learn patterns invariant to different geometric perturbations. Augmentor automates this task by building an oversampling pipeline defined by a series of operations, such as rotating and flipping, to perform on a set of images. This approach allows chaining transformation operations together to create new images[96] and makes Augmentor a popular choice.[97–99]

Our Augmentor pipeline included rotations (−15°–15°), shearing (−15°–15°), skewing (0%–20%), and horizontal flips. Figure 2 displays an illustration of the output of each operation. The pipeline was used to oversample class 2 ($n_{lm \subset train}$ = 761), class 3 ($n_{sls \subset train}$ = 1,416), class 4 ($n_{ts \subset train}$ = 1,123), class 5 ($n_{eb \subset train}$ = 800), class 7 ($n_{mv \subset train}$ = 298), and class 9 ($n_{sm \subset train}$ = 1,340) of the training set to 1,500 images each (Figure 3).

Algorithm 5: KNN

In contrast to oversampling, undersampling techniques remove data from the majority class to balance data distribution. While the easiest technique involves randomly undersampling the majority class, it does not show any concern for the importance of the data being removed, so that useful information can be lost. An alternative approach is to be more selective with the instances being removed by involving learning models that can identify redundant data such as the KNN ML algorithm. KNN calculates feature similarity between its training data to predict values for new, previously unseen data. When given a new input, KNN finds k (a user-predefined number) most resembling data (nearest neighbors) using similarity metrics, such as the Euclidean distance. Based on the majority class of those similar data, the algorithm classifies the input. Despite its simplicity, KNN is considered one of the top 10 data-mining algorithms.[100] When used for undersampling on 33 different datasets, KNN outperformed the results of the edited NN (ENN), neighbor cleaning rule (NCL), and random undersampling algorithms.[101]

Figure 2 shows an example of KNN circling three detected neighbors with similar features belonging to the same class. Here, KNN was used to discover similarities of leaves in class 8 of the training set ($n_{ylcv \subset train}$ = 4,285), removing images with redundant features, ultimately undersampling this majority class to 1,500 images (Figure 3). The input images were first converted to three dimensional arrays with values ranging from 0 to 255 (representing the brightness of each color in the RGB channels). Following Beckmann et al.,[101] for each image, the algorithm deleted the nearest k neighbors (here k = 3). The process repeats until the number of images reaches the undersampling goal (1,500 images).

Step 3: Data analysis

Algorithm 6: Random forest

Random forest (RF) is an EL algorithm introduced in 2001.[102] RF employs multiple decision trees (a forest) to create an ensemble (Figure 2). A decision tree is an ML algorithm used for both classification and regression tasks. It has a tree-like structure with two types of nodes: decision and leaf nodes. Decision nodes represent a test that is applied on the input data, where a decision will be made based on its result. Leaf nodes are the final output of those decisions. They represent the final prediction and do not contain any further branches.

In RF, each decision tree in the ensemble is generated from a random subset of the training set, with replacement (i.e., the same data can be sampled more than once), called the bootstrap sample. Lastly, RF determines the prediction by averaging the outputs of all decision trees for regression tasks, or by using majority voting for classification tasks. When evaluated on 121 different datasets against 17 algorithms, including discriminant analysis, support vector machines, and decision trees, RF proved to be the best classifier among them.[103] Although Wainberg et al.[104] questioned this conclusion, arguing that the results are biased by the lack of a held-out test set, RF is still possibly one of the most famous EL algorithms because of its effectiveness and simplicity.[105]

We used the preprocessed training set to train an RF classifier with 1,000 decision trees to classify tomato leaf diseases and spider mites. We extracted three features from the training dataset to be used as input: (1) Haralick texture[106] to identify texture characteristics of leaves; (2) Hu moments, also known as image moments,[107] to characterize the shape of leaves, including

information such as their area, centroid, and orientation; and (3) color histograms[108] to calculate the distribution of pixel color intensities, which improves classification on the assumption that images from the same class will have similar histograms. Figure 3 shows the resulting confusion matrix for the 10-class classification task.

Algorithm 7: DCNN

Similar to DNNs, DCNNs are made up of layers of artificial neurons that have learnable weights and biases. However, they use convolution operations instead of matrix multiplications between layers, making them more effective in processing imaging data. A DCNN is commonly composed of three main types of layers. The network starts with a convolutional layer, which can be followed by additional convolutional or pooling layers and ends with the fully connected layer. The convolutional layer employs a feature detector, known as a filter or kernel, that moves across an input image and checks for the presence of important features. This operation is known as convolution. A dot product is calculated between the input pixels and the filter, storing the output in a new matrix. Afterward, the filter shifts by a predefined number of pixels, known as stride, repeating the process until it has swept the entire image. The final output is known as feature map or activation map. After each convolution, a nonlinear activation function is applied to the feature map, such as ReLU transformation, which simply outputs the input value directly if it is positive, and outputs zero for all negative inputs. The pooling layer reduces the number of features in representations captured by convolutional layers. Similarly, it sweeps a filter across the entire input, applying an aggregation function that chooses the maximum (maximum pooling) or the average (average pooling) value of the input to populate an output matrix. Finally, the fully connected layer, typically a DNN, performs classification on the features extracted by the series of convolutional and pooling layers. An example architecture of DCNN is illustrated in Figure 2. Features learned at different layers of DCNN correspond to different levels of abstractions. For example, initial layers detect edges and extract color information, and the following layers encode shapes and textures.[109] Such learned features can be transferred from one model to another in a process called transfer learning. Transferring knowledge from previously trained models can be more efficient than training a DCNN from scratch. Among various DL algorithms, DCNNs are the most extensively studied.[110] They have made a series of breakthroughs in image-based analysis[21,54,110] and became the default choice for image classification tasks.[54,111] We trained a standard DCNN using the preprocessed training set. Then, we repeated the process with another DCNN that transfers a densely connected convolutional network (DenseNet)-161 model pretrained on the ImageNet dataset. Both networks followed the architecture of DenseNet-161 described by Huang et al.,[112] used Adam optimizer, and had a learning rate of 0.0001 and a batch size of 20. These hyperparameters can be tuned based on the performance of the validation set. After 100 epochs of training, the best-performing model on the validation set for the standard DCNN was generated at epoch 93, compared with 83 for the pretrained model. Figure 3 shows the confusion matrix for both models generated using the test set, suggesting that the pretrained model not only trained faster but is also a better classifier.

Step 4: Performance analysis and explanation

Once an algorithm is trained, it generates a model that can be used to make new predictions on data that it has not seen before. The test set is therefore used to evaluate the model predictions; they can either be correct or wrong. The ratio of correct predictions to total number of predictions shows how accurate is the model. Intuitively, a good classifier should generate a higher number of correct predictions and thus is expected to have a higher accuracy. However, the accuracy measure does not inform whether classes are being predicted equally well or some classes are performing better than others. Moreover, in the case of imbalanced data, a classifier may achieve an extremely high accuracy by correctly predicting the majority class only.

Alternatively, a confusion matrix describes the complete performance of the model. It summarizes predictions in four categories: true-positive, true-negative, false-positive, and false-negative. It also forms the basis for the other types of metrics, including precision, recall, and F1 score. Figure 3 depicts the confusion matrix for each of the three trained models, cross-tabulating the true labels against the predicted ones. The matrices revealed that, out of the three models, the pretrained DenseNet-161 DCNN had the highest prediction accuracy for all 10 classes.

Algorithm 8: LIME

LIME was released in 2016 with the purpose of opening any ML or DL black box models, making their predictions explainable to humans.[113] To explain imaging data, LIME starts by partitioning an image into multiple segments that share common characteristics such as pixel intensity, known as superpixels (Figure 3). It then generates perturbations by deleting some superpixels in the image and monitors how they affect the prediction of the black box model. Finally, it identifies which areas of the image have been important for classification and, thus, generates the explanation by highlighting them (Figure 3). This shows how crucial the superpixels segmentation is for LIME to generate explanations. By default, LIME uses the quickshift algorithm to segment images. However, there are other approaches that have been tested to improve the quality of explanations. For example, Schallner et al.[114] studied the effects of using different superpixel segmentation algorithms on LIME explanations. Their results suggested that the Compact-Watershed algorithm yielded the best results, introducing an improvement of 45.58% over the default quickshift algorithm. LIME is one of the most popular X-AI post hoc algorithms that can generate explanations for predictions produced by any AI black box models.[115] Its performance turned out to be the best one when compared with Shapley additive explanations (SHAP)[116] and model agnostic supervised local explanations (MAPLE)[117] to explain image-based classifications.[118] We used LIME to explain our three trained models, RF, the standard DCNN, and the pretrained DCNN. Both quickshift and Compact-Watershed algorithms were implemented. Figure 3 shows the explanation of the results of the pretrained DCNN, comparing the two algorithms. Our results are in agreement with findings in Schallner et al.,[114] highlighting more precise coverage of the late blight-diseased area on the tomato leaf using Compact-Watershed. Late blight disease remains one of the most significant threats to tomato agriculture, raising global food security concerns

and causing economic hardship to farmers. While it is often misdiagnosed at its first encounter or not managed timely, its real-time, AI-driven detection and interpretation by researchers and farmers become of paramount importance.

MOVING TO THE CLOUD

Large-scale high-throughput plant phenotyping experiments can produce images in the order of gigabytes,[119] making it crucial to have access to greater computing power. GPUs can dramatically increase training speed thanks to their processing cores initially designed to process visual data such as videos and images. GPUs have thousands of these cores that process data in parallel.[120] Some academic supercomputers offer access to GPUs, but, because of the rapid progress in their technology, they are at risk of becoming outdated.[55]

Cloud computing services offer up-to-date hardware infrastructure that can be accessed over the Internet, eliminating the need for researchers or organizations to set up their own. Commercial services such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure enable the quick setup of a single or a group of servers connected to each other, known as a cluster. Importantly, these servers can be easily upgraded or downgraded based on the need.

Further, emerging services such as Google Colab, originally developed to help disseminate AI education and research, enable code development in a web browser, requiring almost no software to be set up. Colab gives its users the ability to write and execute codes (on central processing units [CPUs] or GPUs), take notes, and upload media such as images all in one place, called notebook, for free, but with varying usage limits. These notebooks are ideal to share well-documented, reproducible data analysis pipelines, promoting transparency and reusability.[121]

The increasing availability of cloud computing has led to an overuse of large amounts of data in a compute-centric paradigm, where data are analyzed by clusters in data centers. This approach requires high amount of data transfer, especially with the increased usage of Internet of Things (IoT) devices in agriculture,[122] which can introduce high latency. Although next-generation wireless network technologies such as 5G aspire to provide higher bandwidth and lower latency,[25,123] further concerns arise from the compute-centric approach, including (1) energy efficiency where data transmission is energy intensive,[124] (2) potential privacy violations caused by insecure data transmission and storage on cloud computers,[125] and (3) inefficient use of storage due to unnecessary data transmission.[126] This is leading to the adoption of a data-centric paradigm, where data processing takes place at the origin of data. However, because data generators such as IoT devices are resource constrained in terms of computational power, and because AI algorithms require large computational resources, a new field known as tiny AI or tiny ML (TinyML) has emerged. TinyML aims to compress AI models to fit on hardware with low computational power so that they can process data locally, reducing the need for data transmission. In a typical TinyML workflow, AI algorithms are trained on normal computers or in the cloud. Then, generated models are compressed using compression techniques such as network pruning, low-rank approximation, weight quantization, and network architecture transform[127] to allow for their deployment in devices with low computational resources, without significant decrease in their performance.[128] This ability empowers IoT and resource-constrained devices with AI capabilities, opening up doors to many new possibilities.

LESSONS AND THE WAY FORWARD

As AI becomes more and more integrated into agriculture, we envisage a new green revolution with digital phenomics at the helm. Digital phenomics is enormously important for developing novel disease control strategies and accelerating breeding of tomato, widely considered one of our most important vegetable crops. We used an existing expert-curated open dataset and publicly available image processing resources to identify and classify nine biologically and economically important diseases and spider mites that can infect leaves of tomato fruit-bearing crops, resulting in yield losses worldwide, and threatening our food supply and safety. In efforts toward embracing open science, the contextualized programming codes and computational notebooks are open source and ready for use immediately in image-based analyses. We believe that code, notebooks, and data sharing along with the availability of open-source programming languages, software frameworks, software libraries, and IDEs are bringing us closer to democratized AI research. Crucially, though, to support this, it is essential to promote an open research culture and to nurture talents for long-term progress of digital phenomics. Thinking across disciplinary boundaries in a time of health and climate crisis, this could better support the many ways in which our research community can contribute to the UN's Sustainable Development Goals (SDGs) and promote ethically responsible research and innovation.[129] However, to make global agriculture more sustainable, smaller farms in developing countries must have free access to these resources and tools as well. Ready, Steady, Go AI is more than just a training and learning tool for driving ongoing and likely future biotechnology innovations; it is also useful for motivating graduate students and early-career researchers to engage in co-produced research studies that create space to work together with people with a shared interest. We advocate for creating perforations in the funnel of learning so that exposure to such skills and insights can be integrated into a full mainstream educational program to increase students' and early career researchers' own confidence and ability to undertake complex phenomics procedures using AI. We have a responsibility to our students to ensure they develop the awareness and skills to engage with experts outside plant sciences, recognizing complementary transferable skills and ways of creative and critical thinking.

EXPERIMENTAL PROCEDURES

Resource availability

Lead contact

Further information and requests for digital resources should be directed to and will be fulfilled by the lead contact, Antoine Harfouche (aharfouche@unitus.it).

Materials availability

This study did not generate new materials.

Data and code availability

The code of this tutorial is available in interactive notebooks runnable on Google Colab and hosted on GitHub at https://github.com/HarfoucheLab/Ready-Steady-Go-AI.

The original PlantVillage dataset is available at https://doi.org/10.17632/tywbtsjrjv.

Our processed data, including cropped and segmented images, and the trained AI models are available on Mendeley Data at https://doi.org/10.17632/4g7k9wptyd.

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.patter.2021.100323.

ACKNOWLEDGMENTS

We would like to thank the editors and the reviewers of our manuscript for their constructive feedback that resulted in a much-improved paper. Partial support for this work was provided by the EU FP7 project WATBIO, grant no. 311929; the EU H2020 project EMPHASIS-PREP and its Italian node, PHEN-ITALY, grant no. 739514; and by the Italian Ministry of University and Research Brain Gain Professorship to A.L.H.

AUTHOR CONTRIBUTIONS

The AI tutorial was conceived and designed by F.N. and A.L.H. Codes and algorithms were written by F.N. Notebooks on Google Colab were created and run by F.N. The manuscript was prepared by F.N. and A.L.H.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

1. Kernighan, B. (1972). A Tutorial Introduction to the Language B (Bell Laboratories).
2. Kernighan, B. (1975). Programming in C - A Tutorial (Bell Laboratories).
3. Kernighan, B., and Ritchie, D. (1978). The C Programming Language (Prentice-Hall, Inc.).
4. McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine, Part I. Commun. ACM *3*, 184–195.
5. Yeung, G.C.N., and Yeung, H.C. (1985). Principles of programming languages: design, evaluation, and implementation. Proc. IEEE *73*, 1245.
6. Newell, A., and Shaw, J.C. (1957). Programming the logic theory machine. In Western Joint Computer Conference: Techniques for Reliability, M. Astrahan, ed. (ACM Press), pp. 230–240.
7. Copeland, J., and Proudfoot, D. (2009). Turing's test. In Parsing the Turing Test, R. Epstein, G. Roberts, and G. Beber, eds. (Springer Netherlands), pp. 119–138.
8. Turing, A.M. (1950). I.—computing machinery and intelligence. Mind *LIX*, 433–460.
9. Newell, A., and Simon, H. (1956). The logic theory machine–A complex information processing system. IEEE Trans. Inf. Theor. *2*, 61–79.
10. Simon, H.A. (1993). Anecdotes–A very early expert system. IEEE Ann. Hist. Comput. *15*, 64–68.
11. Moor, J. (2006). The Dartmouth College artificial intelligence conference: the next fifty years. AI Mag. *27*, 87.
12. Lighthill, J. (1973). Artificial intelligence: a general survey. In Artificial Intelligence: A Paper Symposium (Science Research Council), pp. 1–21.
13. Olazaran, M. (1993). A sociological history of the neural network controversy. In Advances in Computers, C.M. Yovits, ed. (Elsevier), pp. 335–425.
14. Agar, J. (2020). What is science for? The Lighthill report on artificial intelligence reinterpreted. Br. J. Hist. Sci. *53*, 289–310.
15. Zadpoor, A. (2019). Fifty years is not a lot of time!. Matter *1*, 1096–1098.
16. Kaul, V., Enslin, S., and Gross, S.A. (2020). History of artificial intelligence in medicine. Gastrointest. Endosc. *92*, 807–812.
17. Feigenbaum, E.A. (1980). Expert systems: looking back and looking ahead. In GI - 10. Jahrestagung. Informatik-Fachberichte, *33*, R. Wilhelm, ed. (Berlin, Heidelberg: Springer), pp. 1–14.
18. Delipetrev, B., Tsinaraki, C., and Kostić, U. (2020). Historical Evolution of Artificial Intelligence, EUR 30221EN (Luxembourg: Publications Office of the European Union).
19. Feigenbaum, E., and Shrobe, H. (1993). The Japanese national fifth generation project: introduction, survey, and evaluation. Futur. Gener. Comput. Syst. *9*, 105–117.
20. Hassabis, D. (2017). Artificial intelligence: chess match of the century. Nature *544*, 413–414.
21. Krizhevsky, A., Sutskever, I., and Hinton, G.E. (2017). ImageNet classification with deep convolutional neural networks. Commun. ACM *60*, 84–90.
22. Chao, X., Kou, G., Li, T., and Peng, Y. (2018). Jie Ke versus AlphaGo: a ranking approach using decision making method for large-scale data with incomplete information. Eur. J. Oper. Res. *265*, 239–247.
23. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of Go without human knowledge. Nature *550*, 354–359.
24. Cox, M., and Ellsworth, D. (1997). Application-controlled demand paging for out-of-core visualization. In Proceedings. Visualization '97 (Cat. No. 97CB36155), R. Yagel and H. Hagen, eds. (IEEE), pp. 235–244.
25. Harfouche, A.L., Jacobson, D.A., Kainer, D., Romero, J.C., Harfouche, A.H., Scarascia Mugnozza, G., Moshelion, M., Tuskan, G.A., Keurentjes, J.J.B., and Altman, A. (2019). Accelerating climate resilient plant breeding by applying next-generation artificial intelligence. Trends Biotechnol. *37*, 1217–1235.
26. Streich, J., Romero, J., Gazolla, J.G.F.M., Kainer, D., Cliff, A., Prates, E.T., Brown, J.B., Khoury, S., Tuskan, G.A., Garvin, M., et al. (2020). Can exascale computing and explainable artificial intelligence applied to plant biology deliver on the United Nations sustainable development goals? Curr. Opin. Biotechnol. *61*, 217–225.
27. Mann, A. (2020). Core concept: nascent exascale supercomputers offer promise, present challenges. Proc. Natl. Acad. Sci. U S A *117*, 22623–22625.
28. Furbank, R.T., and Tester, M. (2011). Phenomics – technologies to relieve the phenotyping bottleneck. Trends Plant Sci. *16*, 635–644.
29. Yang, W., Feng, H., Zhang, X., Zhang, J., Doonan, J.H., Batchelor, W.D., Xiong, L., and Yan, J. (2020). Crop phenomics and high-throughput phenotyping: past decades, current challenges, and future perspectives. Mol. Plant *13*, 187–214.
30. Tardieu, F., Cabrera-Bosquet, L., Pridmore, T., and Bennett, M. (2017). Plant phenomics, from sensors to knowledge. Curr. Biol. *27*, R770–R783.
31. Xue, B., Sartori, P., and Leibler, S. (2019). Environment-to-phenotype mapping and adaptation strategies in varying environments. Proc. Natl. Acad. Sci. U S A *116*, 13847–13855.
32. Alberch, P. (1991). From genes to phenotype: dynamical systems and evolvability. Genetica *84*, 5–11.
33. Pigliucci, M. (2010). Genotype–phenotype mapping and the end of the "genes as blueprint" metaphor. Philos. Trans. R. Soc. B Biol. Sci. *365*, 557–566.
34. Ahnert, S.E. (2017). Structural properties of genotype–phenotype maps. J. R. Soc. Interface *14*, 20170275.

35. Bzdok, D., Altman, N., and Krzywinski, M. (2018). Statistics versus machine learning. Nat. Methods *15*, 233–234.

36. Singh, A.K., Ganapathysubramanian, B., Sarkar, S., and Singh, A. (2018). Deep learning for plant stress phenotyping: trends and future perspectives. Trends Plant Sci. *23*, 883–898.

37. Singh, A., Jones, S., Ganapathysubramanian, B., Sarkar, S., Mueller, D., Sandhu, K., and Nagasubramanian, K. (2021). Challenges and opportunities in machine-augmented plant stress phenotyping. Trends Plant Sci. *26*, 53–69.

38. van Klompenburg, T., Kassahun, A., and Catal, C. (2020). Crop yield prediction using machine learning: a systematic literature review. Comput. Electron. Agric. *177*, 105709.

39. Chlingaryan, A., Sukkarieh, S., and Whelan, B. (2018). Machine learning approaches for crop yield prediction and nitrogen status estimation in precision agriculture: a review. Comput. Electron. Agric. *151*, 61–69.

40. Orhobor, O.I., Alexandrov, N.N., and King, R.D. (2020). Predicting rice phenotypes with meta and multi-target learning. Mach. Learn. *109*, 2195–2212.

41. Azodi, C.B., Pardo, J., VanBuren, R., de los Campos, G., and Shiu, S.-H. (2020). Transcriptome-based prediction of complex traits in maize. Plant Cell *32*, 139–151.

42. Liu, Y., Wang, D., He, F., Wang, J., Joshi, T., and Xu, D. (2019). Phenotype prediction and genome-wide association study using deep convolutional neural network of soybean. Front. Genet. *10*, 1091.

43. Azodi, C.B., Tang, J., and Shiu, S.-H. (2020). Opening the black box: interpretable machine learning for geneticists. Trends Genet. *36*, 442–455.

44. Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat. Mach. Intell. *1*, 206–215.

45. Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., et al. (2020). Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. Inf. Fusion *58*, 82–115.

46. McCulloch, W.S., and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. *5*, 115–133.

47. Singh, A., Ganapathysubramanian, B., Singh, A.K., and Sarkar, S. (2016). Machine learning for high-throughput stress phenotyping in plants. Trends Plant Sci. *21*, 110–124.

48. McCarthy, J. (1968). Programs with common sense. In Semantic Information Processing, M. Minsky, ed. (MIT Press), pp. 403–418.

49. Deng, C., Ji, X., Rainey, C., Zhang, J., and Lu, W. (2020). Integrating machine learning with human knowledge. iScience *23*, 101656.

50. Muggleton, S. (1991). Inductive logic programming. New Gener. Comput. *8*, 295–318.

51. Muggleton, S., and Marginean, F. (2000). Logic-based machine learning. In Logic-Based Artificial Intelligence, J. Minker, ed. (Springer US), pp. 315–330.

52. Cropper, A., and Morel, R. (2021). Learning programs by learning from failures. Mach. Learn. *110*, 801–856.

53. van Dijk, A.D.J., Kootstra, G., Kruijer, W., and de Ridder, D. (2021). Machine learning in plant science and plant breeding. iScience *24*, 101890.

54. LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature *521*, 436–444.

55. Zou, J., Huss, M., Abid, A., Mohammadi, P., Torkamani, A., and Telenti, A. (2019). A primer on deep learning in genomics. Nat. Genet. *51*, 12–18.

56. Schmidhuber, J. (2015). Deep learning in neural networks: an overview. Neural Networks *61*, 85–117.

57. Fukushima, K. (1980). Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybern. *36*, 193–202.

58. Fukushima, K. (2014). Modeling vision with the neocognitron. In Springer Handbook of Bio-/Neuroinformatics, N. Kasabov, ed. (Springer Berlin Heidelberg), pp. 765–782.

59. Levine, A.B., Schlosser, C., Grewal, J., Coope, R., Jones, S.J.M., and Yip, S. (2019). Rise of the machines: advances in deep learning for cancer diagnosis. Trends Cancer *5*, 157–169.

60. Dietterich, T.G. (2000). Ensemble methods in machine learning. In Multiple Classifier Systems (Springer Berlin Heidelberg), pp. 1–15.

61. Zhou, Zhi-Hua (2009). Ensemble learning. In Encyclopedia of Biometrics, S. Li and A. Jain, eds. (Boston: Springer), pp. 270–273.

62. Schapire, R.E. (1990). The strength of weak learnability. Mach. Learn. *5*, 197–227.

63. Hansen, L.K., and Salamon, P. (1990). Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell. *12*, 993–1001.

64. Breiman, L. (1996). Bagging predictors. Mach. Learn. *24*, 123–140.

65. Yuan, J., Wen, T., Zhang, H., Zhao, M., Penton, C.R., Thomashow, L.S., and Shen, Q. (2020). Predicting disease occurrence with high accuracy based on soil macroecological patterns of *Fusarium* wilt. ISME J. *14*, 2936–2950.

66. Gomez Selvaraj, M., Vergara, A., Montenegro, F., Alonso Ruiz, H., Safari, N., Raymaekers, D., Ocimati, W., Ntamwira, J., Tits, L., Omondi, A.B., et al. (2020). Detection of banana plants and their major diseases through aerial images and machine learning methods: a case study in DR Congo and Republic of Benin. ISPRS J. Photogramm. Remote Sens. *169*, 110–124.

67. Li, Y., Li, M., Li, C., and Liu, Z. (2020). Forest aboveground biomass estimation using Landsat 8 and Sentinel-1A data with machine learning algorithms. Sci. Rep. *10*, 9952.

68. Wang, Y., Wu, G., Deng, L., Tang, Z., Wang, K., Sun, W., and Shangguan, Z. (2017). Prediction of aboveground grassland biomass on the Loess Plateau, China, using a random forest algorithm. Sci. Rep. *7*, 6940.

69. Ansarifar, J., Akhavizadegan, F., and Wang, L. (2020). Performance prediction of crosses in plant breeding through genotype by environment interactions. Sci. Rep. *10*, 11533.

70. Shahhosseini, M., Hu, G., Huber, I., and Archontoulis, S.V. (2021). Coupling machine learning and crop modeling improves crop yield prediction in the US Corn Belt. Sci. Rep. *11*, 1606.

71. Scott, M.L. (2009). Programming Language Pragmatics (Elsevier).

72. Perkel, J.M. (2019). Julia: come for the syntax, stay for the speed. Nature *572*, 141–142.

73. Gao, K., Mei, G., Piccialli, F., Cuomo, S., Tu, J., and Huo, Z. (2020). Julia language in machine learning: algorithms, applications, and open issues. Comput. Sci. Rev. *37*, 100254.

74. Grabowski, P., and Rappsilber, J. (2019). A primer on data analytics in functional genomics: how to move from data to insight? Trends Biochem. Sci. *44*, 21–32.

75. Goluguri, N.V.R.R., Suganya Devi, K., and Vadaparthi, N. (2021). Image classifiers and image deep learning classifiers evolved in detection of *Oryza sativa* diseases: survey. Artif. Intell. Rev. *54*, 359–396.

76. Francesconi, S., Harfouche, A., Maesano, M., and Balestra, G.M. (2021). UAV-based thermal, RGB imaging and gene expression analysis allowed detection of Fusarium head blight and gave new insights into the physiological responses to the disease in durum wheat. Front. Plant Sci. *12*, 551.

77. Maesano, M., Khoury, S., Nakhle, F., Firrincieli, A., Gay, A., Tauro, F., and Harfouche, A. (2020). UAV-based LiDAR for high-throughput determination of plant height and above-ground biomass of the bioenergy grass *Arundo donax*. Remote Sens *12*, 3464.

78. Hughes, D.P., and Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing. CoRR, arXiv:1511.08060.

79. Perez, L., and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. arXiv, arXiv:1712.04621.

80. Fahlgren, N., Feldman, M., Gehan, M.A., Wilson, M.S., Shyu, C., Bryant, D.W., Hill, S.T., McEntee, C.J., Warnasooriya, S.N., Kumar, I., et al. (2015). A versatile phenotyping system and analytics platform reveals diverse temporal responses to water availability in *Setaria*. Mol. Plant *8*, 1520–1535.

81. Gehan, M.A., Fahlgren, N., Abbasi, A., Berry, J.C., Callen, S.T., Chavez, L., Doust, A.N., Feldman, M.J., Gilbert, K.B., Hodge, J.G., et al. (2017). PlantCV v2: image analysis software for high-throughput plant phenotyping. PeerJ *5*, e4088.

82. Knecht, A.C., Campbell, M.T., Caprez, A., Swanson, D.R., and Walia, H. (2016). Image Harvest: an open-source platform for high-throughput plant image processing and analysis. J. Exp. Bot. *67*, 3587–3599.

83. Zhou, S., Chai, X., Yang, Z., Wang, H., Yang, C., and Sun, T. (2021). Maize-IAS: a maize image analysis software using deep learning for high-throughput plant phenotyping. Plant Methods *17*, 48.

84. Lobet, G., Draye, X., and Périlleux, C. (2013). An online database for plant image analysis software tools. Plant Methods *9*, 38.

85. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: unified, real-time object detection. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (IEEE), pp. 779–788.

86. Redmon, J., and Farhadi, A. (2017). YOLO9000: better, faster, stronger. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (IEEE), pp. 6517–6525.

87. Redmon, J., and Farhadi, A. (2018). YOLOv3: an incremental improvement. CoRR, arXiv:1804.02767.

88. Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., et al. (2014). Microsoft COCO: common objects in context. In European Conference on Computer Vision, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds. (Springer), pp. 740–755.

89. Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). SegNet: a deep convolutional encoder-decoder architecture for image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. *39*, 2481–2495.

90. Brostow, G.J., Fauqueur, J., and Cipolla, R. (2009). Semantic object classes in video: a high-definition ground truth database. Pattern Recognit. Lett. *30*, 88–97.

91. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. In Advances in Neural Information Processing Systems, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, eds. (Curran Associates, Inc.), pp. 2672–2680.

92. Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv, arXiv:1511.06434.

93. Chawla, N.V., Bowyer, K.W., Hall, L.O., and Kegelmeyer, W.P. (2002). SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. *16*, 321–357.

94. He, H., Yang, B., Garcia, E.A., and Li, S. (2008). ADASYN: adaptive synthetic sampling approach for imbalanced learning. In 2008 IEEE International Joint Conference on Neural Networks (IEEE), pp. 1322–1328.

95. Sampath, V., Maurtua, I., Aguilar Martín, J.J., and Gutierrez, A. (2021). A survey on generative adversarial networks for imbalance problems in computer vision tasks. J. Big Data *8*, 27.

96. Bloice, M.D., Stocker, C., and Holzinger, A. (2017). Augmentor: an image augmentation library for machine learning. arXiv, arXiv:1708.04680.

97. Abdelhalim, I.S.A., Mohamed, M.F., and Mahdy, Y.B. (2021). Data augmentation for skin lesion using self-attention based progressive generative adversarial network. Expert Syst. Appl. *165*, 113922.

98. Bloice, M.D., Roth, P.M., and Holzinger, A. (2019). Biomedical image augmentation using Augmentor. Bioinformatics *35*, 4522–4524.

99. Jager, P., Moore, G., Calpin, P., Durmishi, X., Salgarella, I., Menage, L., Kita, Y., Wang, Y., Kim, D.W., Blackshaw, S., et al. (2021). Dual midbrain and forebrain origins of thalamic inhibitory interneurons. eLife *10*, e59272.

100. Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., et al. (2008). Top 10 algorithms in data mining. Knowl. Inf. Syst. *14*, 1–37.

101. Beckmann, M., Ebecken, N.F.F., and Pires de Lima, B.S.L. (2015). A KNN undersampling approach for data balancing. J. Intell. Learn. Syst. Appl. *07*, 104–116.

102. Breiman, L. (2001). Random forests. Mach. Learn. *45*, 5–32.

103. Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? J. Mach. Learn. Res. *15*, 3133–3181.

104. Wainberg, M., Alipanahi, B., and Frey, B.J. (2016). Are random forests truly the best classifiers? J. Mach. Learn. Res. *17*, 1–5.

105. González, S., García, S., Del Ser, J., Rokach, L., and Herrera, F. (2020). A practical tutorial on bagging and boosting based ensembles for machine learning: algorithms, software tools, performance study, practical perspectives and opportunities. Inf. Fusion *64*, 205–237.

106. Haralick, R.M., Shanmugam, K., and Dinstein, I. (1973). Textural features for image classification. IEEE Trans. Syst. Man. Cybern. *SMC-3*, 610–621.

107. Hu, Ming-Kuei (1962). Visual pattern recognition by moment invariants. IEEE Trans. Inf. Theor. *8*, 179–187.

108. Chakravarti, R., and Meng, X. (2009). A study of color histogram based image retrieval. In 2009 Sixth International Conference on Information Technology: New Generations, S. Latifi, ed. (IEEE), pp. 1323–1328.

109. Gopalakrishnan, K., Khaitan, S.K., Choudhary, A., and Agrawal, A. (2017). Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection. Constr. Build. Mater. *157*, 322–330.

110. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. (2018). Recent advances in convolutional neural networks. Pattern Recognit *77*, 354–377.

111. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. *115*, 211–252.

112. Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K.Q. (2017). Densely connected convolutional networks (IEEE), pp. 4700–4708.

113. Ribeiro, M.T., Singh, S., and Guestrin, C. (2016). Why should I trust you? In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ACM), pp. 1135–1144.

114. Schallner, L., Rabold, J., Scholz, O., and Schmid, U. (2019). Effect of superpixel aggregation on explanations in LIME - a case study with biological data. In Machine Learning and Knowledge Discovery in Databases International Workshops of ECML PKDD, P. Cellier and K. Driessens, eds. (Springer), pp. 147–158.

115. Linardatos, P., Papastefanopoulos, V., and Kotsiantis, S. (2020). Explainable AI: a review of machine learning interpretability methods. Entropy *23*, 18.

116. Lundberg, S.M., and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Advances in Neural Information Processing Systems, U. V Luxburg, I. Guyon, S. Bengio, H. Wallach, and R. Fergus, eds. (Curran Associates, Inc.), pp. 4768–4777.

117. Plumb, G., Molitor, D., and Talwalkar, A.S. (2018). Model agnostic supervised local explanations. In Advances in Neural Information Processing Systems, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, and N. Cesa-Bianchi, eds. (Curran Associates, Inc.), pp. 2520–2529.

118. Guidotti, R. (2021). Evaluating local explanation methods on ground truth. Artif. Intell. *291*, 103428.

119. Roitsch, T., Cabrera-Bosquet, L., Fournier, A., Ghamkhar, K., Jiménez-Berni, J., Pinto, F., and Ober, E.S. (2019). Review: new sensors and data-driven approaches—a path to next generation phenomics. Plant Sci. *282*, 2–10.

120. Bhattacharya, A., and Cui, Y. (2017). A GPU-accelerated algorithm for bi-clustering analysis and detection of condition-dependent coexpression network modules. Sci. Rep. *7*, 4162.

121. Clarke, D.J.B., Jeon, M., Stein, D.J., Moiseyev, N., Kropiwnicki, E., Dai, C., Xie, Z., Wojciechowicz, M.L., Litz, S., Hom, J., et al. (2021). Appyters: turning Jupyter notebooks into data-driven web apps. Patterns *2*, 100213.

122. Tzounis, A., Katsoulas, N., Bartzanas, T., and Kittas, C. (2017). Internet of things in agriculture, recent advances and future challenges. Biosyst. Eng. *164*, 31–48.

123. Hassan, N., Yau, K.-L.A., and Wu, C. (2019). Edge computing in 5G: a review. IEEE Access *7*, 127276–127289.

124. Aslan, J., Mayers, K., Koomey, J.G., and France, C. (2018). Electricity intensity of internet data transmission: untangling the estimates. J. Ind. Ecol. *22*, 785–798.

125. vurukonda, N., and Rao, B.T. (2016). A study on data storage security issues in cloud computing. Proced. Comput. Sci. *92*, 128–135.

126. Brous, P., Janssen, M., and Herder, P. (2020). The dual effects of the Internet of Things (IoT): a systematic review of the benefits and risks of IoT adoption by organizations. Int. J. Inf. Manage. *51*, 101952.

127. Li, B., Chen, P., Liu, H., Guo, W., Cao, X., Du, J., Zhao, C., and Zhang, J. (2021). Random sketch learning for deep neural networks in edge computing. Nat. Comput. Sci. *1*, 221–228.

128. Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2020). A survey of model compression and acceleration for deep neural networks. arXiv, arXiv:1710.09282.

129. Harfouche, A.L., Petousi, V., Meilan, R., Sweet, J., Twardowski, T., and Altman, A. (2021). Promoting ethically responsible use of agricultural biotechnology. Trends Plant Sci. *26*, 546–559.

About the Authors

Farid Nakhle is currently a PhD candidate in computational phenomics at Antoine Harfouche's laboratory at the University of Tuscia. He earned his MSc in computer science at the American University of Science and Technology and has extensive experience in computer networks, programming and artificial intelligence, and DevOps engineering gained over his 7-year career in software and web development. Farid assisted in designing and successfully delivering a practical tutorial on machine learning and ensemble learning to PhD students and early career researchers in accelerating climate-resilient plant breeding by applying omics and artificial intelligence at the Swedish University of Agricultural Sciences.

Antoine Harfouche is an associate professor (Brain Gain Program, Italian Ministry of University and Research) of plant biotechnology at the University of Tuscia. His interdisciplinary research aims to connect the dots between phenomics, genomics, artificial intelligence, and plant breeding. He publishes in these fields. He is the chair of the multidisciplinary research program of the Information and Communication Technologies in Organizations and Society (ICTO), France. He is a member of the management committee of the Italian Plant Phenotyping Network (PHEN-ITALY—national node of ESFRI EU project EMPHASIS-PREP—European infrastructure for multi-scale plant phenotyping and simulation for food security in a changing climate). He co-organized, lectured, and tutored an accredited PhD course unit in omics data analytics and AI in accelerating climate resilient plant breeding by applying omics and artificial intelligence at the Swedish University of Agricultural Sciences. Dr. Harfouche is passionate about open science, enabling FAIRness (findability, accessibility, interoperability, and reusability) of computational omics, and promoting ethically responsible research and innovation in biotechnology and beyond.