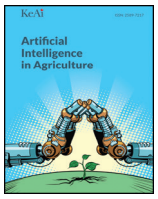
Artificial Intelligence in Agriculture 6 (2022) 55–67

Contents lists available at ScienceDirect

Artificial Intelligence in Agriculture

journal homepage: http://www.keaipublishing.com/en/journals/artificial-intelligence-in-agriculture/



# Few-shot learning for biotic stress classification of coffee leaves

Lucas M. Tassis [b], Renato A. Krohling [a,b,*]

[a] LABCIN - Nature Inspired Computing Lab, Federal University of Espirito Santo, Av. Fernando Ferrari, 514, CEP, 29075-910 Vitória, Espırito Santo, ES, Brazil
[b] PPGI - Graduate Program in Computer Science, Federal University of Espirito Santo, Av. Fernando Ferrari, 514, CEP 29075-910, Vitória, Espírito Santo, ES, Brazil

ARTICLE INFO

*Article history:*
Received 17 August 2021
Received in revised form 31 December 2021
Accepted 1 April 2022
Available online 9 April 2022

*Keywords:*
Plant diseases and pests classification
Image classification
Few-shot learning
Meta-learning
Convolutional neural networks

ABSTRACT

In the last few years, deep neural networks have achieved promising results in several fields. However, one of the main limitations of these methods is the need for large-scale datasets to properly generalize. Few-shot learning methods emerged as an attempt to solve this shortcoming. Among the few-shot learning methods, there is a class of methods known as embedding learning or metric learning. These methods tackle the classification problem by learning to compare, needing fewer training data. One of the main problems in plant diseases and pests recognition is the lack of large public datasets available. Due to this difficulty, the field emerges as an intriguing application to evaluate the few-shot learning methods. The field is also relevant due to the social and economic importance of agriculture in several countries. In this work, datasets consisting of biotic stresses in coffee leaves are used as a case study to evaluate the performance of few-shot learning in classification and severity estimation tasks. We achieved competitive results compared with the ones reported in the literature in the classification task, with accuracy values close to 96%. Furthermore, we achieved superior results in the severity estimation task, obtaining 6.74% greater accuracy than the baseline.


## 1. Introduction

Plants are often exposed to environmental stresses that can cause a vast impact on growth and production. Plant stresses can be biotic and abiotic. Biotic stresses are caused by pathogens such as fungi, bacteria, viruses. They can also be caused by pests. On the other hand, abiotic stresses are caused by heat, drought, cold, salinity, among others (Suzuki et al., 2014).

Coffee is one of the main crops in Brazil, and particularly, in the state of Espírito Santo. One of the main challenges faced by farmers is the pests and diseases that affect the coffee crops. They can limit the yield for both small and big farmers, causing losses that may impair the crop harvest. Each stress requires precise management, and the use of the wrong technique may cause damage to the environment and the worker's health. They can also decrease the quality of the final product (Ventura et al., 2017).

Among the biotic stresses that affect coffee leaves, four of the main ones are leaf miner, rust, brown leaf spot, and cercospora leaf spot. Fig. 1 illustrates symptoms caused by the stresses.

In addition to classifying the stress that affects the leaves, it is also important to estimate its severity. The severity is measured by the percentage of leaf area that is injured. Precise severity estimation can assist in predicting yield loss, monitoring and forecasting epidemics, and correct management decisions (Bock et al., 2010).

Classifying plant biotic stresses and estimating their severity is a complex task due to several factors: lack of precise delimitation of the lesion area; variation of characteristics between lesions of the same type (such as color, shape, and size); the presence of multiples lesions on the same leaf; and the fact that different stresses may have similar symptoms (Barbedo, 2016). In the last few years, deep learning techniques have been presenting promising results in image classification and recognition (Goodfellow et al., 2016). These techniques have even been used successfully in the classification of plant diseases (Boulent et al., 2019), emerging as a useful tool to assist farmers and agriculture professionals.

However, traditional machine learning and deep learning techniques present some limitations, among them is the need for large-scale datasets to the models generalize well (Wang et al., 2020). So, new methods were proposed aiming to create models that generalized well with fewer data. This led to the proposal of *meta-learning methods* (Hospedales et al., 2020), and in particular, the *few-shot learning methods*.

The limited number of datasets is one of the main challenges that influence the use of deep learning techniques for diseases and pests recognition in plants (Barbedo, 2018). This leads to weak models since the datasets do not have enough examples for deep neural

* Corresponding author at: LABCIN - Nature Inspired Computing Lab, Federal University of Espirito Santo, Av. Fernando Ferrari, 514, CEP, 29075-910 Vitória, Espırito Santo, ES, Brazil.
*E-mail address:* rkrohling@inf.ufes.br (R.A. Krohling).

https://doi.org/10.1016/j.aiia.2022.04.001

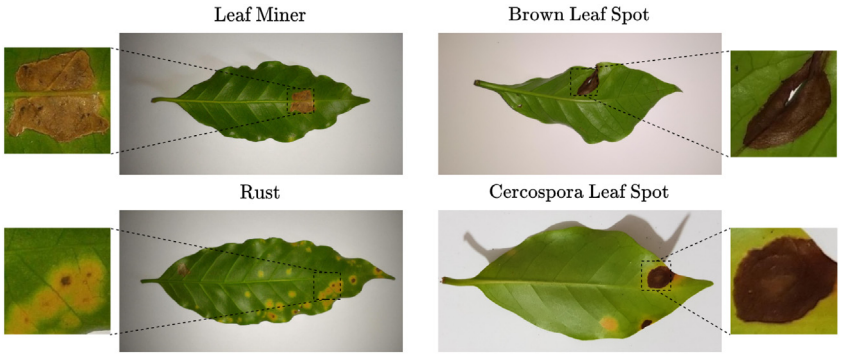



**Fig. 1.** Example of biotic stresses that affect the coffee leaves.

networks to properly generalize. Thereby, the investigation of few-shot learning methods for classifying stresses on plants becomes relevant.

As pointed out, there are several works in the literature that applied deep neural networks for plant disease recognition. Most of them make use of traditional convolutional neural networks in different crops, obtaining accuracy values over 90% (Mohanty et al., 2016; Rahman et al., 2018; Elhassouny and Smarandache, 2019; Geetharamani and Pandian, 2019; Barbedo, 2019).

Some works also explored the use of deep neural networks specifically in coffee crops. Manso et al. (2019) developed a method combining traditional segmentation algorithms with neural networks to classify spots in images of coffee leaves. The algorithm also estimated the severity of the lesions using the segmented spot area. Esgario et al. (2020) used deep learning to classify and estimate the severity of lesions in coffee leaves and symptoms images. In their work, they also evaluated the use of a multi-task network for both classification and severity estimation, comparing with a network trained for a single task. The work used the same datasets used in this work, serving as a baseline for the results obtained using the few-shot learning method. Esgario et al. (2022) developed a similar method, but instead of using traditional segmentation methods, they used deep neural networks in the segmentation task. In addition to the algorithm, an app for smartphones was developed to assist farmers in the identification of diseases and pests. Tassis et al. (2021) proposed a method combining instance and semantic segmentation to identify pests and diseases from in-field coffee trees images. Their approach consisted of a three-stage method that combined instance segmentation, semantic segmentation, and classification to classify and estimate the severity of the lesions affecting the coffee trees.

Recently, some works started exploring the use of few-shot learning in plant disease recognition. Wang and Wang (2019) explored the siamese network architecture to classifiy diseases of three different types of leaves. They obtained accuracy over 90% while using only 20 samples per class in the training set. Argüeso et al. (2020) evaluated siamese networks with triplet loss in the PlantVillage dataset also reaching accuracy values over 90%. The first step was training a convolutional neural network (CNN) for general plant features extraction. Next, they evaluated the fine-tuning for new unseen classes, obtaining similar performance while training on a dataset reduced by almost 90%. Jadon (2020) proposed the SSM (Stacked Siamese Matching) network, combining two few-shot learning methods: Siamese Networks and Matching Networks. The method obtained over 92% accuracy in two plant datasets, outperforming a classical CNN architecture. Next, Afifi et al. (2021) evaluated the performance of Triplet Network and deep Adversarial Metric Learning (DAML) while shifting training domains. They achieved 99% accuracy when the domain shift was small, and 81% when the shift was large. This work also indicates that a model trained on a large dataset can be fine-tuned on small datasets by the use of few-shot learning methods.

In this work, we explore the use of few-shot learning and feature extraction for plant stress recognition and severity estimation. To experiment, we will use a dataset of biotic stresses in coffee leaves. The remainder of this paper is organized as follows. Section 2 presents some fundamental concepts on deep learning and few-shot learning. Moreover, we also describe the methods Triplet Network and Prototypical Network used in this work. Section 3 presents the experimental setup and results. Finally, in Section 4, we draw some conclusions.

## 2. Method

### 2.1. Preliminaries

In this section, we present some fundamental concepts on deep learning, describing the convolutional neural networks.

Convolutional neural networks (CNN) (LeCun et al., 1989) were proposed to process data with grid-like topology such as images and has had success in practical applications in computer vision (Goodfellow et al., 2016). A standard CNN is composed of three main types of layers: *convolutional*, *pooling* and *fully connected layers*. Fig. 2 illustrates an example of a generic CNN architecture.

#### 2.1.1. Convolutional layer

The network is named *convolutional* after the mathematical operation of *convolution* applied in this layer (Goodfellow et al., 2016). The layer is responsible for extracting features from the input data. These features can be edges, points, or even full objects. The convolutional layer consists of a set of learnable filters, also known as *kernels*. These kernels slide (*i.e.* convolve) across the input data producing a *feature map* as an output (also known as *activation maps*) (Li et al., 2021). At the end of a convolution layer, usually, we apply an *activation function* in the resulting feature map. Activation functions are important because they introduce non-linearity into the output, remove all negative values, and highlight important features. A usual activation function used in the literature is the rectified linear unit, or simply, ReLU (Goodfellow et al., 2016).

#### 2.1.2. Pooling layer

In the pooling layer, the feature maps from the convolution layer are spatially reduced by a pooling (also known as downsampling) operation. The pooling operation statistically summarizes regions from the activation maps, reducing their size while maintaining their main features (Goodfellow et al., 2016). Two common ways of performing the pooling operation are maximum and average polling. In the maximum pooling, we summarize the set of values by its higher value. On the other hand, in the average pooling, we summarize the set of values by calculating the average value.

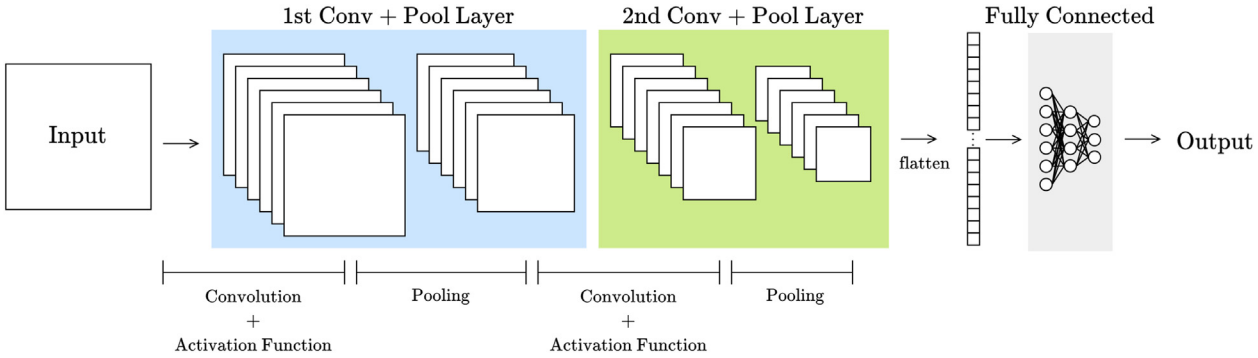



**Fig. 2.** Illustration of a convolutional neural network consisting of convolutional, pooling and fully connected layers.

*2.1.3. Fully connected layer*

The fully connected layer consists of a typical feedforward network, also known as multilayer perceptrons (MLPs) (Goodfellow et al., 2016). They usually act as the classifier in the CNNs, using the features extracted by the previous layers as input. In the context of CNNs, the inputs fed to a fully connected network are the feature maps from the convolution and pooling layers. Since the feature maps are tensors and the fully connected network receives a vector as input, a *flatten* operator is applied to the maps, returning a *vector*.

A CNN architecture can also be used without a fully connected layer, returning the flattened vector as output. This vector is sometimes referred to as an *embedding*. This kind of architecture that produces embeddings is extensively used in the few-shot learning context, as presented in the next section.

In the following, we present some fundamental concepts on few-shot learning. First, Section 2.2 presents some basic concepts and definitions in few-shot learning. Next, Section 2.3 presents the embedding learning methods, describing the ones used in this work.

*2.2. Few-shot learning*

One of the main shortcomings in machine and deep learning is the need of large-scale data to properly generalize (Wang et al., 2020). To address this issue, the meta-learning paradigm was proposed (Hospedales et al., 2020), and, in particular, the *few-shot learning methods* (Wang et al., 2020).

Few-shot learning (*FSL*) aims to solve tasks using a limited amount of training data. This class of algorithms was inspired by the fact that humans only need a small number of examples to learn and generalize some tasks. For instance, given a few photos of a stranger, we can easily recognize this person on another set of pictures (Wang et al., 2020).

Most *FSL* methods proposed are for supervised learning problems such as image classification and object recognition. Although not explored in this work, one of the main uses of few-shot learning is learning to generalize to new classes unseen during training (Wang et al., 2020; Hospedales et al., 2020; Chen et al., 2020).

Unlike in most machine and deep learning conventional algorithms, in the few-shot learning context (and more particularly, in the few-shot image classification context), we usually train the models on a set of *tasks* created from the training data. Some authors also refer to this as *episodes* (Ravi and Larochelle, 2017).

Each task has a *support set* and a *query set*. The query set consists of the data that we want to make some inference about (*i.e.* data that we want to classify in a classification problem). The support set consists of the labeled data that we use to assist in the inference. We usually denote a task as a $n$-way $k$-shot problem, where $n$ is the number of classes in the support set; and $k$ is the number of examples per class in the support set. Fig. 3 illustrates an example of a few-shot learning task.

There are a few approaches in the literature to categorize the *FSL* methods. One of them is organizing the methods in initialization based methods, distance metric learning based methods, and hallucination based methods (Chen et al., 2020).

*2.2.1. Initialization based methods*

Address the few-shot learning problem by *learning to fine tune*. There are two main approaches to this. One of them attempts to learn good model initialization (*i.e.* parameters of the network) and the other focuses on learning the optimizer. Both approaches diminish the need for data or the number of gradient steps to train the network.

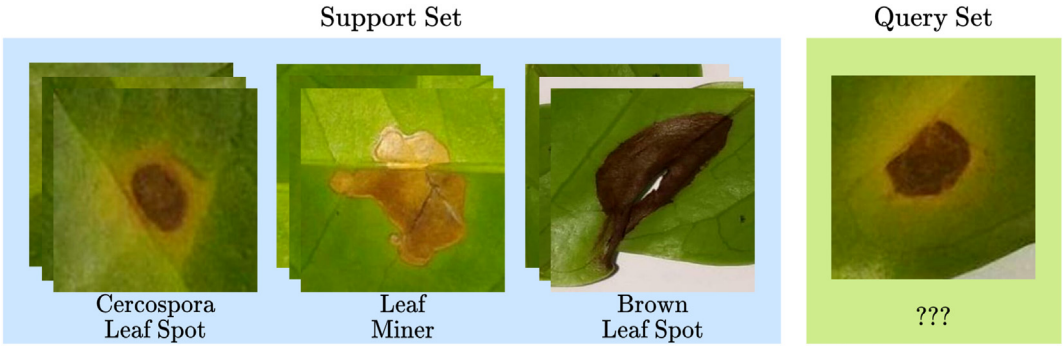



**Fig. 3.** Example of a 3-way 3-shot task. In this example, $n = 3$ because we have three different classes (cercospora leaf spot, leaf miner, and brown leaf spot) in the support set; and $k = 3$ because we have three examples per class in the support set. The query set contains the image that we want to classify assisted by the ones in the support set.

*2.2.2. Distance metric learning based methods*

Address the few-shot learning problem by *learning to compare*. These methods attempt to decrease the need for data by training a model to learn the similarity between images, instead of training the model to simply classify the input image. Intuitively, if a model can distinguish two different images, it can learn to classify unseen classes with few data. These methods are also known as embedding learning methods.

*2.2.3. Hallucination based methods*

Address the few-shot learning problem by *learning to augment*. These methods tackle the lack of data directly, attempting to learn a generator using the data available. This generator can then create new instances for data augmentation.

In our work, we focus on distance metric learning-based, also known as embedding learning, methods. This class of methods was chosen due to the promising results in image classification tasks. In the next section, we provide more insight and definitions about this class of methods.

*2.3. Embedding learning*

Embedding learning is a class of few-shot learning methods that address the classification task as a comparison problem by *learning to compare*. But how do the models learn to compare? The embedding learning methods produce a representation (*i.e.* an embedding) of the input data with lower dimensionality than the original input using an embedding function. Next, these embeddings are compared by a similarity function, in a way that embeddings from the same class are close together, and embeddings from different classes are separated (Wang et al., 2020). Fig. 4 illustrates a generic embedding learning model.

Specifically, in the image classification task, the embedding function is usually a CNN without the fully connected layer, since the flattened vector is a lower-dimensional representation of the input data. We may also refer to the embedding function as feature extractor. The similarity function is usually a distance function like the Euclidean or cosine distance, depending on the model used.

There are many embedding learning models proposed in the literature. Among them, perhaps the most known are: Siamese Network (Koch et al., 2015), Triplet Network (Schroff et al., 2015), Matching Networks (Vinyals et al., 2016), Prototypical Network (Snell et al., 2017), and Relation Network (Sung et al., 2018). In our work, we only used the Triplet Network and Prototypical Network. The Prototypical Network was selected because, despite being a relatively straightforward method, it still achieves competitive performance when compared to more modern methods. The Triplet Network was selected because it is one of the first few-shot learning methods proposed, serving as a baseline for the few-shot learning results. Both models are described in the following.

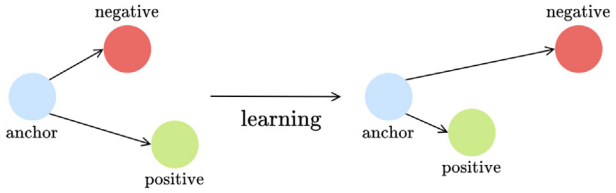



**Fig. 5.** Illustration of the learning process using the triplet loss. The loss aim to minimize the distance between anchor and positive, whilst maximizing the distance between the anchor and negative. Based on the figure by Schroff et al. (2015).

*2.3.1. Triplet network*

The Triplet Network (TripletNet) (Schroff et al., 2015) consists of a few-shot learning model that uses three images as input (thus the name *triplet*): *anchor*, *positive*, and *negative*. The anchor consists of the image that we want to classify. Next, the positive consists of an image that is from the same class as the anchor. Lastly, the negative consists of an image from another class (that is not the anchor's class). The objective is to produce a representation of the inputs, so that the anchor is similar to the positive and different from the negative.

Since we are working with images, the TripletNet uses a CNN model as its feature extractor (embedding function). The similarity function used is the squared Euclidean distance. The model aims to generalize in a way that the embeddings of the anchor and the positive are close to each other in the feature space (are similar), and the embeddings from the anchor and negative are far away from each other (are different). To that end, the TripletNet uses a loss function named *triplet loss*, described by Eq. 1 (Schroff et al., 2015):

$$L(x_a, x_p, x_n) = \max\left(\|f(x_a) - f(x_p)\|_2^2 - \|f(x_a) - f(x_n)\|_2^2 + \alpha 0\right) \quad (1)$$

where, $f(x_a)$ is the embedded anchor, $f(x_p)$ is the embedded positive, $f(x_n)$ is the embedded negative, and $\alpha$ is the margin enforced between positive and negative pairs.

The idea behind the triplet loss can be visualized in Fig. 5. Fig. 6 illustrates the TripletNet model. Algorithm 1 presents pseudo-code for the loss computation in a single batch.

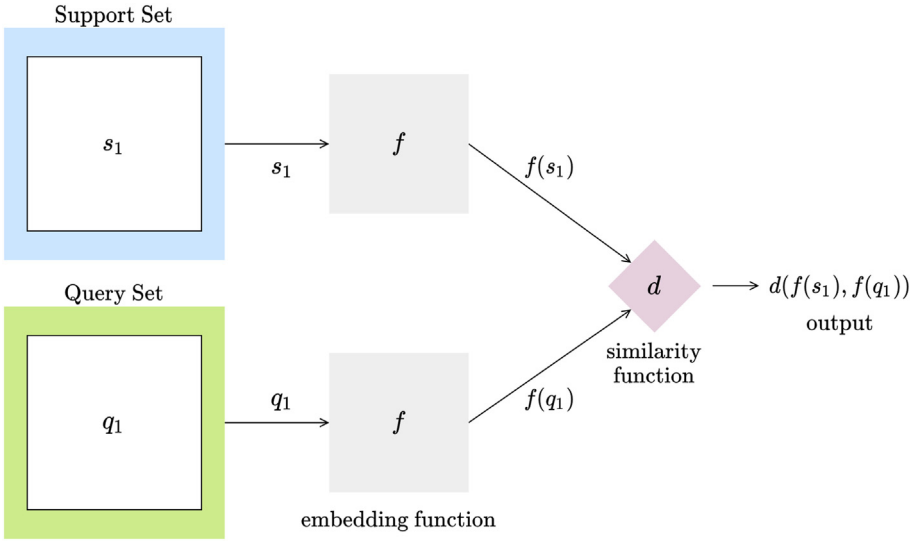



**Fig. 4.** Generic embedding learning model illustration. The inputs images are used as input for an embedding function. The embeddings produced are compared by a similarity function, resulting in the output.

**Algorithm 1:** Loss computation for a single batch using TripletNet. $N$ is the number of classes in the training set. $\texttt{RandomSample}(S, n)$ denotes a set of $n$ elements chosen uniformly at random from a set $S$. The function $f(x)$ denotes the embedding function.

**Input:** Training set $D = \{(x_1, y_1), \ldots, (x_d, y_d)\}$, where $y_i \in 1, \ldots, N$. $D_n$ denotes the subset of $D$ containing all elements $(x_i, y_i)$ such that $y_i = n$.

**Output:** The triplet loss $\mathcal{L}$.

$x_a, y_a \leftarrow \texttt{RandomSample}(D, 1)$ ; // Randomly select an anchor.

$x_p, y_p \leftarrow \texttt{RandomSample}(D_{y_a}, 1)$ ; // Randomly select a positive.

$x_n, y_n \leftarrow \texttt{RandomSample}(D - D_{y_a}, 1)$ ; // Randomly select a negative.

$\mathcal{L} \leftarrow \max(\|f(x_a) - f(x_p)\|_2^2 - \|f(x_a) - f(x_n)\|_2^2 + \alpha, 0)$ ; // Compute the loss.

**return** $\mathcal{L}$

At the end of the training stage, the TripletNet with triplet loss produces a feature extractor used to cluster instances from the same class. In order to use it as a classifier, we usually combine this model with a *k*-nearest neighbors classifier (KNN) (Cunningham and Delany, 2020), in a way that the embeddings from the training set act as the training samples to the KNN.

### 2.3.2. Prototypical networks

The Prototypical Network (ProtoNet) (Snell et al., 2017) is one of the most effective approaches among the few-shot learning methods. The main idea is that instead of comparing the embeddings from the support set directly with the query set embedding, the ProtoNet computes a *prototype* for each class in the support set, and compares the query embedding with them.

Formally, for each class $n$ in the support set, we compute the prototype $c_n = \frac{1}{k}\sum_{i=1}^{k} f(x_i)$, where $k$ is the number of shots, and $f(x_i)$ is the embedding for each example $x_i$ from class $n$. After computing the prototypes, the embedded query sample $f(\mathbf{x})$ is compared with the prototypes using a distance function. Unlike the TripletNet, the ProtoNet can produce an output without the addition of another classifier. Given the distance function $d$, the ProtoNet can produce a distribution over classes for a query $\mathbf{x}$ based on a softmax over distances to the prototypes (Snell et al., 2017):

$$p(y = n|\mathbf{x}) = \frac{\exp\left(-d\left(f(\mathbf{x}), c_n\right)\right)}{\sum_{n'} \exp\left(-d\left(f(\mathbf{x}), c_{n'}\right)\right)} \tag{2}$$

where $\mathbf{x}$ is the query input data, $y$ is the corresponding label, $d$ is the distance function (in our work we use the Euclidean distance), $f(\mathbf{x})$ is the embedded query input, and $c_n$ is the prototype for class $n$.

Fig. 7 illustrates the Prototypical Network model. Algorithm 2 presents the loss computation for a single task using a single query example.

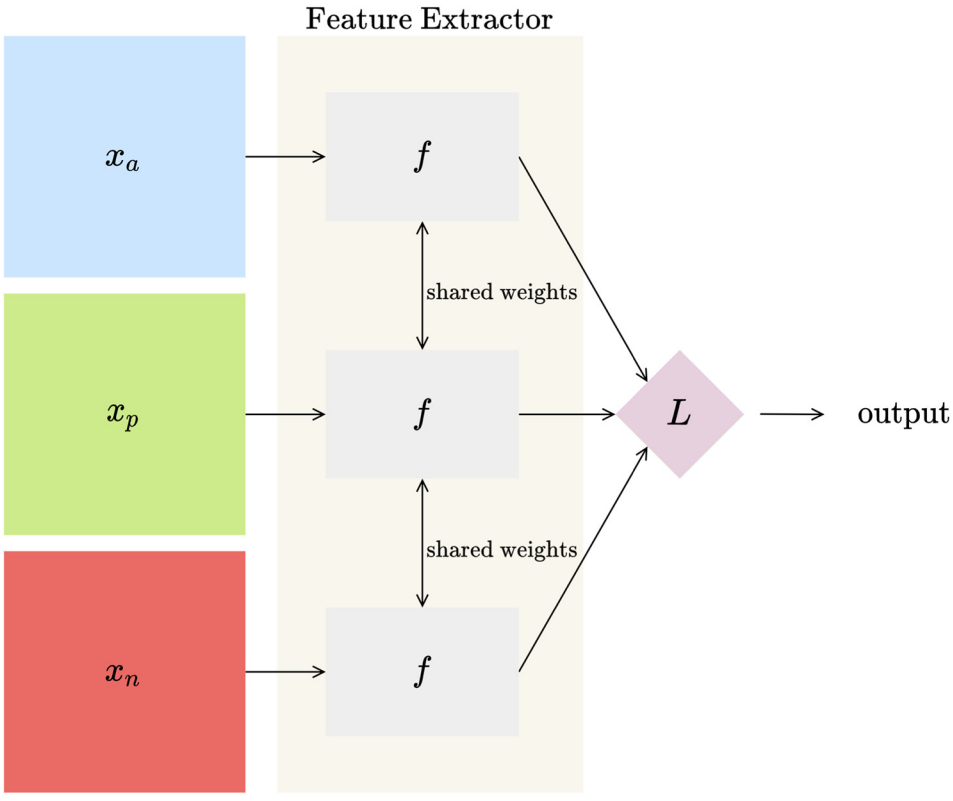



**Fig. 6.** Illustration of the Triplet Network model. The three input images are used as input to the feature extractor. Next, the embeddings are used as input in the triplet loss computation, returning the model's output.

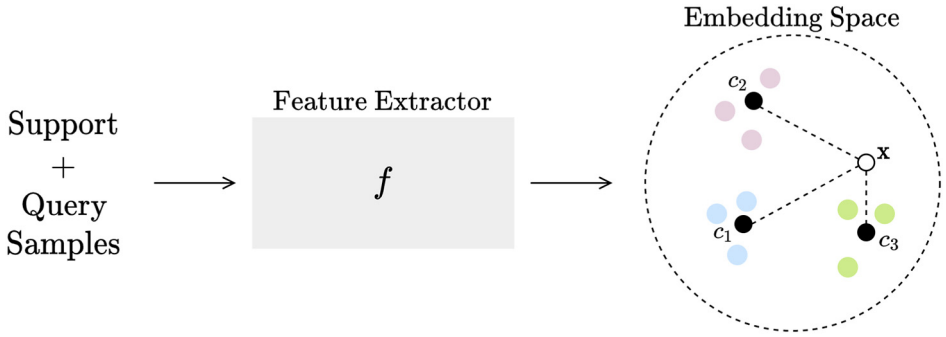



**Fig. 7.** Illustration of the Prototypical Network model. The examples in the query and support set are used as input by a feature extractor. Using the embedded support samples, the prototype for each class in the support set are computed, resulting in $c_1$, $c_2$, and $c_3$. Next, the distances between the embedded query input and prototypes are computed. Lastly, the softmax over the distances results in the classification output.

**Algorithm 2:** Loss computation for a single task with a single query example using ProtoNet. $K$ is the number of examples per class (shots) in the support set. $N$ is the number of classes in the training set. $N_c \leq N$ is the number of classes (ways) in the support set. $\texttt{RandomSample}(S, n)$ denotes a set of $n$ elements chosen uniformly at random from set $S$. The term $\texttt{loss}(x, c)$ is the loss function given the query embedding $x$ and the prototype $c$. The function $f(x)$ denotes the embedding function.

**Input:** Training set $D = \{(x_1, y_1), \ldots, (x_d, y_d)\}$, where $y_i \in 1, \ldots, N$. $D_n$ denotes the subset of $D$ containing all elements $(x_i, y_i)$ such that $y_i = n$.

**Output:** The loss $\mathcal{L}$ for the task.

$\mathbf{x}, y \leftarrow \texttt{RandomSample}(D, 1)$ ; // Select a query point at random.

**for** $n$ *in* $N_c$ **do**

  $S_n \leftarrow \texttt{RandomSample}(D_n, K)$ ; // Select $K$ examples from class $n$

  $c_n \leftarrow \frac{1}{K} \sum_{i \in S_n} f(x_i)$ ; // Computes the prototype for class $n$

**end**

$\mathcal{L} \leftarrow 0$ ; // Initialize the loss

**for** $n$ *in* $N_c$ **do**

  $\mathcal{L} \leftarrow \mathcal{L} + \texttt{loss}(f(\mathbf{x}), c_n)$ ; // For each prototype compute the loss.

**end**

**return** $\mathcal{L}$

## 3. Experiments and results

In this section, we present and diwescuss the experimental results obtained by the few-shot learning methods Triplet Network (Schroff et al., 2015) and Prototypical Network (Snell et al., 2017) in the context of plant stresses classification and severity estimation. Besides using the TripletNet, and ProtoNet we also evaluated several backbones (feature extractors) in each method to assess the contrast among different feature extractors. First, Section 3.1 presents the datasets used in the experiments. Next, Section 3.2 presents the experimental setup used. Section 3.3 presents and discusses the results obtained by the methods used in the experiments.

### 3.1. Dataset

The datasets used in this work were developed by Krohling et al. (2019). The data consists of images of arabica coffee leaves affected by common biotic stresses. The images were captured using smartphones. The datasets are divided in two sets: *Leaf Dataset* and *Symptoms Dataset*. The data used is available for download in GitHub.[1]

[1] www.github.com/esgario/lara2018

#### 3.1.1. Leaf dataset

Consists of a set of 1685 images containing the entire leaf area over a white area background. Each image also contains labels indicating the predominant stress and its severity. There are four biotic stress classes: leaf miner, rust, brown leaf spot, and cercospora leaf spot, in addition to the healthy class, totaling five classes. The stress severity label is divided into five classes, according to the percentage of injured area: healthy (<0.1%), very low (0.1 % − 5%), low (5.1 % − 10%), high (10.1 % − 15%) and very high (>15%). Fig. 8 shows examples of images in this dataset. Table 1 presents some details about the number of instances per class.

#### 3.1.2. Symptoms dataset

Consists of a set of 2722 images from isolated symptoms. Most of the symptoms were cropped from the original leaf images, totaling a number of 2147 symptoms images. In addition to that, Esgario et al. (2020) also used 575 cropped images made available by Barbedo (2019), totaling the 2722 images. Similar to the Leaf Dataset, there are four biotic stress classes: leaf miner, rust, brown leaf spot, and cercospora leaf spot, in addition to the healthy class. Fig. 9 shows examples of images in this dataset. Table 2 presents some details about the number of instances per class.


**Fig. 8.** Examples of leaf dataset images.

**Table 1**
Leaf dataset details.

| Biotic stress | #Images | Severity | #Images |
|---|---|---|---|
| Healthy | 272 | Healthy | 272 |
| Leaf Miner | 387 | Very Low | 924 |
| Rust | 531 | Low | 332 |
| Brown Leaf Spot | 348 | High | 101 |
| Cercospora Leaf Spot | 147 | Very High | 56 |
| Total | 1685 | Total | 1685 |

### 3.2. Experimental setup

The experiments performed were divided in three: Experiment I, Experiment II, and Experiment III. Experiment I consists of the biotic stress classification in the Leaf Dataset. Experiment II consists of the biotic stress classification in the Symptoms Dataset. Finally, Experiment III consists of the severity estimation in the Leaf Dataset.

In each experiment, we evaluated the following few-shot learning models: TripletNet and ProtoNet. In particular, the ProtoNet was evaluated in the 5-way 1-shot and 5-way 5-shot settings. The TripletNet also used a $k$-Nearest Neighbors algorithm[2] as classifier, using the features extracted by the network as input. For each few-shot learning model we evaluated five different backbones (feature extractors) for feature extraction: ResNet50 (He et al., 2016), MobileNetv2 (Sandler et al., 2018), VGG16 (Simonyan and Zisserman, 2014), DenseNet121 (Huang et al., 2017), and EfficientNet-B4 (Tan and Le, 2019). Table 3 contains the number of parameters in each model.

We used a 70%/15%/15% split of the dataset for training, validation, and testing, respectively. The images were resized to 224x224 and the models were pre-trained in ImageNet. Random augmentations (resize, flip, crop, color jitter) were applied online during the training phase. The models trained on 100 epochs, using an SGD optimizer (Ruder, 2016) with learning rate equals to 0.01 (decaying by 0.5 every 20 epochs), momentum equals 0.9, and weight decay equals 0.0005. For each image in the dataset, we randomly generated a support set with images from the training set.

In order to evaluate the models' performance we computed the following metrics: accuracy (ACC), precision (PR), recall (RE), and $F_1$-score ($F_1$). Since the metrics precision and recall are only defined for binary classification, we computed the macro average over the five classes. We also computed the average training time per epoch.

The experiments were carried out in the Google Colab[3] environment with 16GB RAM and a Tesla T4 GPU. All the code was written in Python, with the help of several open-source libraries: PyTorch (Paszke et al., 2019), Learn2Learn (Arnold et al., 2020), NumPy (Harris et al., 2020), Seaborn (Waskom, 2021), Matplotlib (Hunter, 2007), and Scikit-learn (Pedregosa et al., 2011). The source code used in these experiments is available in GitHub.[4]

### 3.3. Results and discussion

In this section, we present the results obtained by the few-shot learning methods in each of the experiments previously described.

#### 3.3.1. Experiment I

**Results**

In the first experiment, we present the results obtained in the biotic stress classification task in the Leaf Dataset. First, we show the results obtained by the TripletNet for each backbone. Table 4 presents the performance metrics obtained by the TripletNet, and Table 5 contains the average training time per epoch.

Next, we present the results obtained by the ProtoNet in the 5-way 1-shot and 5-way 5-shot settings. Table 6 presents the performance metrics obtained by both ProtoNet settings, and Table 7 contains the average training time per epoch.

To ease the visualization of the accuracy per class obtained by the few-shot learning methods, we also provide the confusion matrices for the TripletNet, 5-way 1-shot ProtoNet, and 5-way 5-shot ProtoNet in Fig. 10. We only show the confusion matrices obtained by the backbones with the highest accuracy for each method.

To visualize the models' output embeddings we also provide the scatter plots for TripletNet, 5-way 1-shot ProtoNet, and 5-way 5-shot ProtoNet in Fig. 11. These plots were obtained by reducing the test fold output embeddings into a two-dimensional space using the t-Distributed Stochastic Neighbor Embedding (t-SNE) (Van der Maaten and Hinton, 2008). The t-SNE parameters used to produce the visualization were the default values in the scikit-learn library.[5]

**Discussion**

Analyzing the TripletNet results in Table 4, we observe that all backbones achieved a high accuracy, ranging from 93.25% to 95.24%. The best overall backbones were the MobileNetv2 and VGG16, with an accuracy of 95.24%. However, the MobileNetv2 had a better recall and $F_1$ score than the VGG16. Looking at the training time results in Table 5, we

[2] We tested values of $k = 5,10,15$ and since the results were similar, we only show the results for $k = 5$

[3] colab.research.google.com

[4] www.github.com/lucastassis/pg-coffee

[5] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html


**Fig. 9.** Examples of symptoms dataset images.

also observe that the MobileNetv2 had the shortest training time, with an average training time per epoch of 40.03*s* per epoch, compared to 67.83*s* obtained by the VGG16.

In the results obtained by the ProtoNet, presented in Table 6, we observe that this method had a slightly higher average accuracy when compared to the TripletNet. The best backbone in the 5-way 1-shot setting was the ResNet50 (although the VGG16 and DenseNet121 obtained the same accuracy, the remaining metrics were worse than the ones presented by ResNet50), with an accuracy of 95.63%. As for the 5-way 5-shot setting, the MobileNetv2 obtained the best results in every metric, with an accuracy of 96.03%. Regarding training times, the MobileNetv2 obtained the shortest, with 14.29*s* and 28.24*s* per epoch in the 5-way 1-shot and 5-way 5-shot settings, respectively.

Analyzing the confusion matrices in Fig. 10, we observe that all the models obtained 100% accuracy in detecting leaves with brown leaf spot. The results obtained for the classes healthy, leaf miner, and brown leaf spot were also high. The lowest accuracy was obtained in the class cercospora leaf spot, with an accuracy value of 67% in all few-shot learning methods. This result coincides with the work by Esgario et al. (2020), in which the lowest accuracy was obtained in this same class. This result may be justified due to the similarity of this lesion with the others or due to the imbalanced dataset. One difference from the work by Esgario et al. (2020), was that they obtained 100% accuracy in healthy leaves, whereas we obtained in the class brown leaf spot. Investigating this, it seems that since the input image resizing causes some distortions in the biotic stresses, and the few-shot learning methods are comparison-based methods, some rust lesions might have become too small and indistinguishable from a healthy leaf.

The scatter plots presented in Fig. 11 regarding the t-SNE embedding visualization shows that the backbones were able to generalize well, with some mistakes concerning the cercospora leaf spot class, that had a higher intersection with the other classes' clusters.

The best result reported by Esgario et al. (2020) in the biotic stress classification task using the Leaf Dataset was 95.63%, using a ResNet50 as classifier. This result is similar to ours, and the 5-way 5-shot ProtoNet with MobileNetv2 backbone even slightly outperformed it, with an accuracy of 96.03%.

### 3.3.2. Experiment II

**Results**

In the second experiment, we present the results obtained in the biotic stress classification task in the Symptoms Dataset. First, we show the results obtained by the TripletNet for each backbone. Table 8 presents the performance metrics obtained by the TripletNet, and Table 9 contains the average training time per epoch.

Next, we present the results obtained by the ProtoNet in the 5-way 1-shot and 5-way 5-shot settings. Table 10 presents the performance metrics obtained by both ProtoNet settings, and Table 11 contains the average training time per epoch.

To ease the visualization of the accuracy per class obtained by the few-shot learning methods, we also provide the confusion matrices for the TripletNet, 5-way 1-shot ProtoNet, and 5-way 5-shot ProtoNet in Fig. 12. We only show the confusion matrices obtained by the backbones with the highest accuracy for each method.

To visualize the models' output embeddings we also provide the scatter plots for TripletNet, 5-way 1-shot ProtoNet, and 5-way 5-shot ProtoNet in Fig. 13.

**Table 2**
Symptoms dataset details.

| Biotic stress | #Images |
|---|---|
| Healthy | 256 |
| Leaf Miner | 593 |
| Rust | 991 |
| Brown Leaf Spot | 504 |
| Cercospora Leaf Spot | 378 |
| Total | 2722 |

**Table 3**
Number of parameters (M) in each model.

| Backbone | Parameters (M) |
|---|---|
| ResNet50 | 25.56 |
| MobileNetv2 | 3.4 |
| VGG16 | 138.36 |
| DenseNet121 | 7.97 |
| EfficientNet-B4 | 19 |

**Table 4**
Performance metrics obtained by Triplet Networks in Experiment I.

| TripletNet results | | | | |
|---|---|---|---|---|
| Backbone | ACC (%) | PR (%) | RE (%) | $F_1$ (%) |
| ResNet50 | 94.05 | 92.21 | 88.96 | 90.17 |
| MobileNetv2 | 95.24 | 94.62 | 91.45 | 92.71 |
| VGG16 | 95.24 | 95.12 | 90.69 | 92.25 |
| DenseNet121 | 93.25 | 92.23 | 87.14 | 88.64 |
| EfficientNet-B4 | 93.65 | 91.73 | 87.49 | 88.85 |

**Discussion**

Observing the TripletNet results in Table 8, we may note that again the methods obtained a high accuracy, ranging from 94.63% to 96.42%. In particular, the best result was obtained by the MobileNetv2 backbone, with an accuracy of 96.42%, precision of 96.24, recall of 96.54%, and $F_1$-score of 96.38%. The best average time per epoch was also from MobileNetv2. Other backbones such as the ResNet50, DenseNet121, and EfficientNet-B4 also obtained close results to the one by MobileNetv2. The VGG16 had the worst scores out of all the backbones.

Analyzing the ProtoNet results in Table 10 we observe that all the backbones had a good performance with a best accuracy of 96.72% in the 5-way 1-shot setting and 96.42% in the 5-way 5-shot. Although it is expected that a higher number of shots will translate into higher accuracy, we obtained a higher average accuracy in the 1-shot setting. We understand that this is due to the fact that since the Symptoms Dataset contains only a small number of classes, the feature extractor does not need many support examples per class to generalize well. It also indicates that a single sample from each class contains all the representative features from the class. Considering only the results obtained in the 5-way 1-shot setting, we observe that the EfficientNet-B4 outscored every backbone in every metric, with an accuracy of 96.72%, precision of 95.73%, recall of 96.91% and F1-Score of 96.72%. However, looking the results in Table 11, we observe that this backbone had the largest training time, with an average of 37.68*s* per epoch. In comparison, the ResNet50 and MobileNetv2 obtained accuracies of 96.12% and 96.42% in 21.28*s* and 16.28*s* per epoch, respectively. These accuracies are competitive with the ones obtained by EfficientNet-B4, but the models were almost twice as fast to train. In particular, the MobileNetv2 had a great trade-off in terms of accuracy and average time per epoch. The DenseNet121 and VGG16 obtained lower results of accuracy and also had a higher training time than the ResNet50 and MobileNetv2. Considering the results obtained in the 5-way 5-shot setting, we observe that in this setting the ResNet50 had the best performance. The backbone obtained an accuracy of 96.42%, precision of 96.24%, recall of 96.27%, and F1-Score of 96.25%. The second and third best accuracies were obtained by the EfficientNet-B4 (96.11%) and MobileNetv2 (95.82%), respectively. Similar to the results presented for the 1-shot setting, the MobileNetv2 and ResNet50 had the best trade-off between accuracy and training time.

Analyzing the confusion matrices in Fig. 12, we observe that all the models obtained 100% accuracy in classifying healthy leaves. The lowest accuracy was obtained in the class cercospora leaf spot, with accuracy values ranging from 90% to 94%. The lower accuracy on the cercospora leaf spot class coincides with the previous experiment's results. One interesting thing to note is that the results obtained in the biotic stress classification task were better in the Symptoms Dataset than the ones obtained in the Leaf Dataset. This is in agreement with those provided by Esgario et al. (2020) and Barbedo (2019). Since the images in the Symptoms Datasets focus only on the lesion itself, the network has an easier task of detecting the region of interest. For example, in the Leaf Dataset, the networks have to deal with background and non-injured areas in the leaf, which makes the classification task harder.

The scatter plots presented in Fig. 11 regarding the t-SNE embedding visualization shows that the backbones were able to generalize well. Note that in this experiment the clusters have less intersection than the ones presented in Experiment I, corroborating with the better classification scores.

The best result reported by Esgario et al. (2020) in the biotic stress classification task using the Symptoms Dataset was 97.07%, using a ResNet50 as a classifier. This result is similar to ours (96.72%). This difference is about 0.35% and it may be due training variation.

**Table 6**
Performance metrics obtained by Prototypical Networks in Experiment I.

| ProtoNet Results | | | | |
|---|---|---|---|---|
| 5-way 1-shot | | | | |
| Backbone | ACC (%) | PR (%) | RE (%) | $F_1$ (%) |
| ResNet50 | 95.63 | 95.72 | 91.94 | 93.36 |
| MobileNetv2 | 94.44 | 92.32 | 91.36 | 91.80 |
| VGG16 | 95.63 | 95.07 | 91.64 | 93.02 |
| DenseNet121 | 95.63 | 94.74 | 91.94 | 93.04 |
| EfficientNet-B4 | 94.44 | 92.53 | 89.54 | 90.63 |
| 5-way 5-shot | | | | |
| Backbone | ACC (%) | PR (%) | RE (%) | $F_1$ (%) |
| ResNet50 | 94.84 | 93.38 | 91.25 | 92.14 |
| MobileNetv2 | 96.03 | 96.12 | 92.21 | 93.70 |
| VGG16 | 94.44 | 92.95 | 90.24 | 91.31 |
| DenseNet121 | 95.63 | 95.72 | 91.87 | 93.32 |
| EfficientNet-B4 | 95.24 | 94.00 | 92.05 | 92.89 |

*3.3.3. Experiment III*

**Results**

In the third experiment, we present the results obtained in the severity estimation task in the Leaf Dataset. First, we show the results obtained by the TripletNet for each backbone. Table 12 presents the performance metrics obtained by the TripletNet. Since this experiment uses the Leaf Dataset, the average training time per epoch results are similar to Experiment I. Thus, the average training time per epoch is the same as in Table 5 and Table 7 for TripletNet and ProtoNet, respectively.

Next, we present the results obtained by the ProtoNet in the 5-way 1-shot and 5-way 5-shot settings. Table 13 contains the performance metrics obtained by both ProtoNet settings.

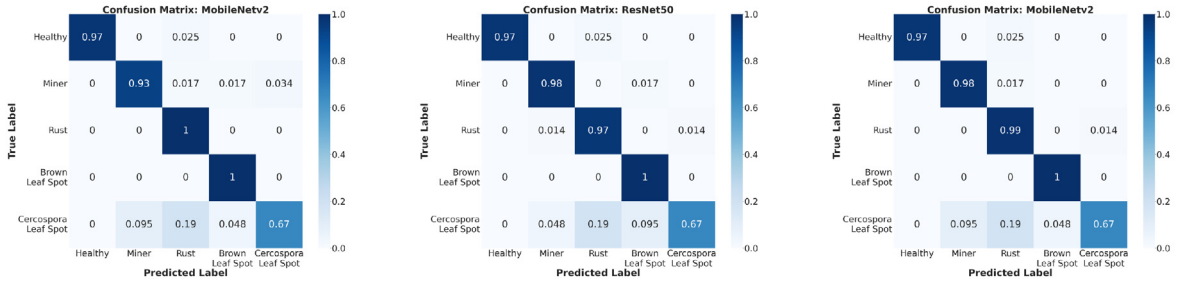
To ease the visualization of the accuracy per class obtained by the few-shot learning methods, we also provide the confusion matrices for the TripletNet, 5-way 1-shot ProtoNet, and 5-way 5-shot ProtoNet in Fig. 14. We only show the confusion matrices obtained by the backbones with the highest accuracy for each method.

**Table 5**
Training time per epoch for Triplet Networks in Experiment I.

| TripletNet Training Time per Epoch (s) | |
|---|---|
| Backbone | Time (s) |
| ResNet50 | 52.97 |
| MobileNetv2 | 40.03 |
| VGG16 | 67.83 |
| DenseNet121 | 70.07 |
| EfficientNet-B4 | 101.87 |

**Table 7**
Training time per epoch for protypical networks in Experiment I.

| ProtoNet training time per Epoch (s) | | |
|---|---|---|
| Backbone | 5-way 1-shot | 5-way 5-shot |
| ResNet50 | 20.75 | 35.43 |
| MobileNetv2 | 14.29 | 28.24 |
| VGG16 | 34.65 | 62.24 |
| DenseNet121 | 28.23 | 59.71 |
| EfficientNet-B4 | 41.52 | 87.28 |




(a) TripletNet with MobileNetv2 Backbone (b) 5-way 1-shot ProtoNet with ResNet50 Backbone (c) 5-way 5-shot ProtoNet with MobileNetv2 Backbone

**Fig. 10.** Confusion matrices obtained by the few-shot learning methods in Experiment I.

To visualize the models' output embeddings we also provide the scatter plots for TripletNet, 5-way 1-shot ProtoNet, and 5-way 5-shot ProtoNet in Fig. 15.

**Discussion**

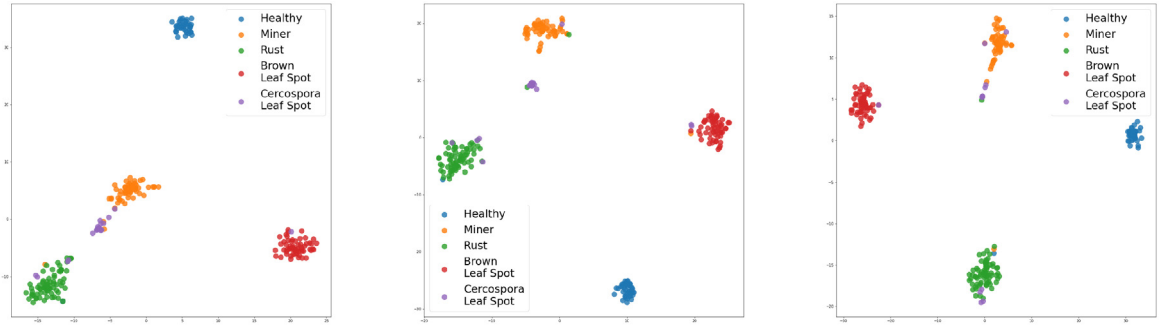
Analyzing the TripletNet results presented in Table 12, we observe that the average accuracy in the severity estimation task was lower than the ones in the previous experiments, with accuracy ranging from 86.11% to 91.27%. The best results were obtained by the MobileNetv2, with an accuracy of 91.27%, precision of 88.25%, recall of 86.57%, and $F_1$-score of 87.32%. The MobileNetv2 also had a lower training time per epoch. The lower scores in precision, recall, and $F_1$-score also indicate that there was a problem in the generalization of some classes. This is expected due to the imbalanced dataset. Also, since there is a hierarchy-like nature between the classes, many of the mistakes are, for example, classifying the severity as *High* instead of *Very High*, which may not be a big mistake. This is observable in the confusion matrices shown. Regarding the results obtained by the ProtoNet in Table 13, we may observe that the 5-way 1-shot setting obtained a higher accuracy than the 5-way 5-shot, with scores of 88.49% and 85.87%, respectively. The 5-way 1-shot setting also obtained higher precision, recall and $F_1$-score. This is an interesting result since as mentioned previously, the 5-way 5-shot setting is expected to perform better than the 5-way 1-shot setting. However, we believe that because of the hierarchy-like nature of the severity classes, when the ProtoNet computes each class prototype (particularly, in the 5-way 5-shot, by computing an average of five embeddings for each class), there may be some distortion in the resulting prototype embedding. Since the difference between classes is not as apparent as it may be in other cases (such as in the biotic stress classification), this average may confuse the network. Considering the 5-way 1-shot setting, we observe that the MobileNetv2 surpassed every model, with an accuracy of 93.25%, precision of 94.11%, recall of 92.40% and $F_1$-score of 93.03%. For the 5-way 5-shot setting, the best performance was obtained by the DenseNet121, with an accuracy of 91.27%, precision of 87.82%, recall of 85.98%, and $F_1$-score of 86.62%.

**Table 8**
Performance metrics obtained by Triplet Networks in Experiment II.

| TripletNet Results | | | | |
|---|---|---|---|---|
| Backbone | ACC (%) | PR (%) | RE (%) | $F_1$ (%) |
| ResNet50 | 95.82 | 95.53 | 95.98 | 95.73 |
| MobileNetv2 | 96.42 | 96.24 | 96.54 | 96.38 |
| VGG16 | 94.63 | 94.53 | 94.41 | 94.46 |
| DenseNet121 | 95.52 | 95.28 | 95.23 | 95.25 |
| EfficientNet-B4 | 95.22 | 95.18 | 95.10 | 95.14 |

**Table 9**
Training time per epoch for Triplet Networks in Experiment II.

| TripletNet Training Time per Epoch (s) | |
|---|---|
| Backbone | Time (s) |
| ResNet50 | 67.03 |
| MobileNetv2 | 50.48 |
| VGG16 | 84.97 |
| DenseNet121 | 88.88 |
| EfficientNet-B4 | 125.82 |




(a) TripletNet with MobileNetv2 Backbone (b) 5-way 1-shot ProtoNet with ResNet50 Backbone (c) 5-way 5-shot ProtoNet with MobileNetv2 Backbone

**Fig. 11.** Visualization obtained by t-SNE for the few-shot learning methods in Experiment I.

**Table 10**
Performance metrics obtained by Prototypical Networks in Experiment II.

| ProtoNet Results | | | | |
|---|---|---|---|---|
| 5-way 1-shot | | | | |
| Backbone | ACC (%) | PR (%) | RE(%) | $F_1$ (%) |
| ResNet50 | 96.12 | 95.91 | 96.14 | 96.02 |
| MobileNetv2 | 96.42 | 96.33 | 96.27 | 96.29 |
| VGG16 | 94.33 | 94.35 | 93.91 | 94.09 |
| DenseNet121 | 95.52 | 95.51 | 95.31 | 95.41 |
| EfficientNet-B4 | 96.72 | 96.66 | 96.75 | 96.70 |
| 5-way 5-shot | | | | |
| Backbone | ACC (%) | PR (%) | RE (%) | $F_1$ (%) |
| ResNet50 | 96.42 | 96.24 | 96.27 | 96.25 |
| MobileNetv2 | 95.82 | 95.77 | 95.63 | 95.68 |
| VGG16 | 94.93 | 94.70 | 94.76 | 94.71 |
| DenseNet121 | 94.93 | 95.10 | 94.68 | 94.84 |
| EfficientNet-B4 | 96.12 | 96.09 | 96.11 | 96.10 |

**Table 11**
Training time per epoch for Triplet Networks in Experiment II.

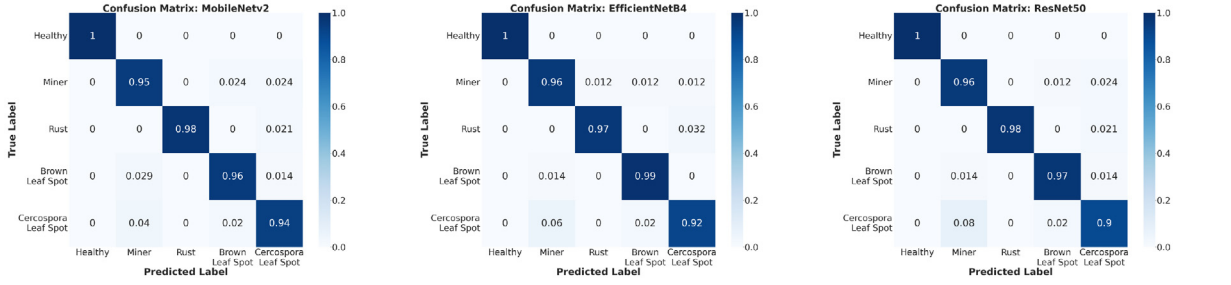
| ProtoNet Training Time per Epoch (s) | | |
|---|---|---|
| Backbone | 5-way 1-shot | 5-way 5-shot |
| ResNet50 | 21.28 | 48.76 |
| MobileNetv2 | 16.28 | 40.93 |
| VGG16 | 35.32 | 78.29 |
| DenseNet121 | 33.34 | 76.29 |
| EfficientNet-B4 | 37.68 | 83.27 |

**Table 12**
Performance metrics obtained by Triplet Networks in Experiment III.

| TripletNet Results | | | | |
|---|---|---|---|---|
| Backbone | ACC (%) | PR (%) | RE (%) | $F_1$ (%) |
| ResNet50 | 86.11 | 83.74 | 80.12 | 81.70 |
| MobileNetv2 | 91.27 | 88.25 | 86.57 | 87.35 |
| VGG16 | 87.30 | 83.77 | 81.10 | 82.37 |
| DenseNet121 | 89.68 | 88.84 | 86.30 | 87.34 |
| EfficientNet-B4 | 86.11 | 83.25 | 75.97 | 78.17 |

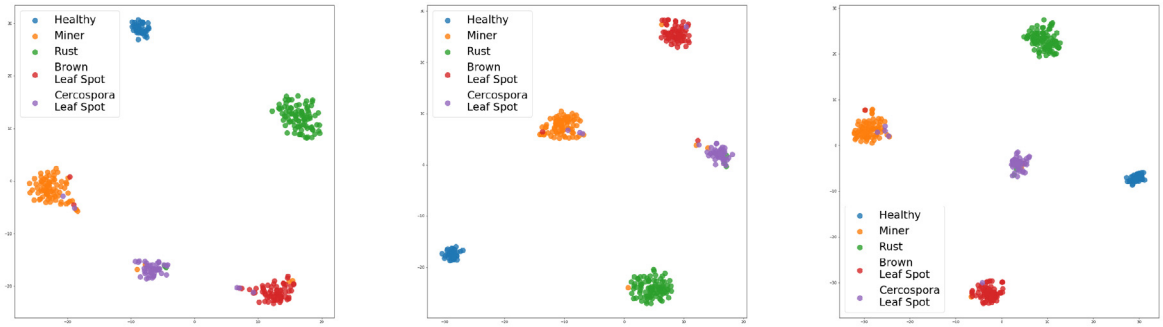
**Table 13**
Performance metrics obtained by Prototypical Networks in Experiment III.

| ProtoNet Results | | | | |
|---|---|---|---|---|
| 5-way 1-shot | | | | |
| Backbone | ACC (%) | PR (%) | RE (%) | $F_1$ (%) |
| ResNet50 | 88.89 | 88.62 | 84.82 | 86.31 |
| MobileNetv2 | 93.25 | 94.11 | 92.40 | 93.03 |
| VGG16 | 84.52 | 74.01 | 74.47 | 74.03 |
| DenseNet121 | 85.32 | 77.18 | 78.00 | 77.09 |
| EfficientNet-B4 | 90.48 | 84.34 | 82.24 | 82.82 |
| 5-way 5-shot | | | | |
| Backbone | ACC (%) | PR (%) | RE (%) | $F_1$ (%) |
| ResNet50 | 83.73 | 74.00 | 73.52 | 73.43 |
| MobileNetv2 | 86.51 | 70.04 | 70.06 | 69.64 |
| VGG16 | 81.35 | 76.36 | 70.63 | 72.63 |
| DenseNet121 | 91.27 | 87.82 | 85.98 | 86.62 |
| EfficientNet-B4 | 86.51 | 82.35 | 81.28 | 81.70 |



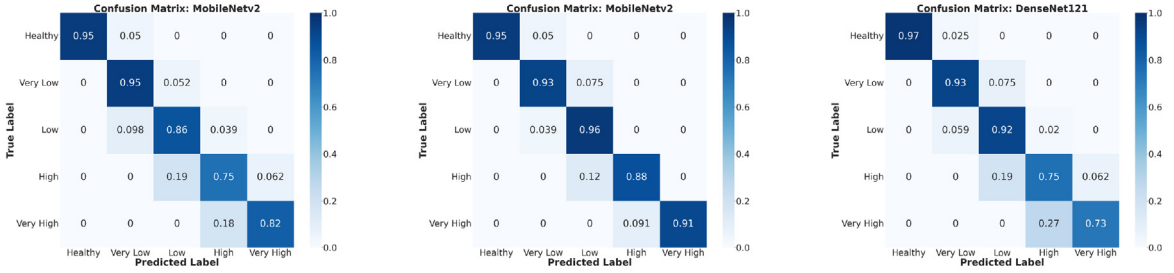

(a) TripletNet with MobileNetv2 Backbone (b) 5-way 1-shot ProtoNet with EfficientNetB4 Backbone (c) 5-way 5-shot ProtoNet with ResNet50 Backbone

**Fig. 12.** Confusion matrices obtained by the few-shot learning methods in Experiment II.



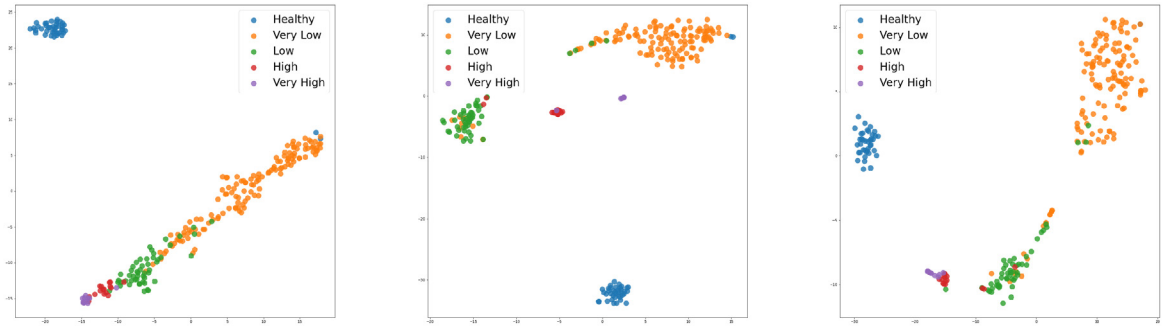

(a) TripletNet with MobileNetv2 Backbone (b) 5-way 1-shot ProtoNet with EfficientNetB4 Backbone (c) 5-way 5-shot ProtoNet with ResNet50 Backbone

**Fig. 13.** Visualization obtained by t-SNE for the few-shot learning methods in Experiment II.




(a) TripletNet with MobileNetv2 Backbone (b) 5-way 1-shot ProtoNet with MobileNetv2 Backbone (c) 5-way 5-shot ProtoNet with DenseNet121 Backbone

**Fig. 14.** Confusion matrices obtained by the few-shot learning methods in Experiment III.




(a) TripletNet with MobileNetv2 Backbone (b) 5-way 1-shot ProtoNet with MobileNetv2 Backbone (c) 5-way 5-shot ProtoNet with DenseNet121 Backbone

**Fig. 15.** Visualization obtained by t-SNE for the few-shot learning methods in Experiment III.

Analyzing the confusion matrices in Fig. 14, we observe that the lowest accuracy was obtained in the class Very High, with accuracy values ranging from 73% to 91%. The lower accuracy on the Very High class may be due to the dataset imbalance. As mentioned before, observing the confusion matrices we note that the mistakes made by the networks are mainly close to the diagonal. This means that the mistakes are not as worrisome as they may seem to be, since most of them may be because of the hierarchy-like nature of the classes.

The scatter plots presented in Fig. 15 regarding the t-SNE embedding visualization show that the methods had more difficulties generalizing the classes. Note that in this experiment the clusters have a higher intersection, particularly between the classes Low, Very Low, High, and Very High, than the ones presented in Experiment I and II.

The best result reported by Esgario et al. (2020) in the severity estimation task using the Leaf Dataset was 86.51%, using a ResNet50 as a classifier. In this particular task, the few-shot learning methods achieved a better result, with an accuracy of 93.25%.

*3.3.4. Overall considerations*

Overall the ProtoNet (in both settings) obtained higher metrics scores than the TripletNet. The 5-way 5-shot was expected to achieve higher accuracy than the 5-way 1-shot methods, however, the 5-way 1-shot had a better training time/accuracy trade-off in most cases. Comparing the backbones, the MobileNetv2 presented the best trade-off between training time and accuracy, outperforming or presenting a similar performance in most cases. This is interesting because, due to its size (see Table 3), the MobileNetv2 can be embedded on smartphones. The results in Experiment I and II were similar to the ones by Esgario et al. (2020), however, the few-shot learning methods had a much better performance in Experiment III. This result shows that few-shot learning methods are promising and might help improve accuracy in the severity estimation task.

## 4. Conclusion

In this work, we evaluated the use of few-shot learning methods applied to plant biotic stress recognition by using Prototypical Networks and Triplet Networks for coffee leaves biotic stress classification and severity estimation. We evaluated the Triplet Networks and Prototypical Networks (both settings) on three experiments, obtaining promising results. In the first experiment, consisting of the biotic stress classification in the Leaf Dataset, we achieved an accuracy of 96.03%. In the second experiment, consisting of the biotic stress classification in the Symptoms Dataset, we achieved an accuracy of 96.72%. Both results were similar to the ones reported in the literature. However, in the third experiment, we achieved an accuracy of 93.25% in the severity estimation task. This result was significantly better than the result reported in the literature and shows promise in the use of few-shot learning methods for plant biotic stress recognition. The work also presented some limitations, among them, there was a lack of performance evaluation of the methods in unseen classes during training. We also did not evaluate any kind of domain shift to assess the adaptability and robustness of the methods to new classes. However, despite the limitations, the achieved results showed great promise in the few-shot learning methods as an alternative for plant biotic stress recognition. Although the work focused on plant biotic stress recognition, the methodology can be expanded to other fields. Future works include the performance evaluation of few-shot learning methods in unseen

classes, evaluation of domain shift to new classes, use of other few-shot learning methods, and decreasing the size of the dataset to assess the amount of data needed for proper generalization.

## Credit author statement

L. M. Tassis- Methodologies, Software, Experiment Simulations, Writing Original Draft, Writing-Reviewing and Editing. R. A. Krohling- Conceptualization, Methodologies, Review, Editing and Supervision.

## Declaration of Competing Interest

All authors disclose that there is no conflict of interest including any financial, personal or other relationships with other people or organizations.

## Acknowledgements

R.A. Krohling thanks the Brazilian research agency Conselho Nacional de Desenvolvimento Científico e Tecnólogico (CNPq) for financial support under grant no. 304688/2021-5.

## References

Afifi, A., Alhumam, A., Abdelwahab, A., 2021. Convolutional neural network for automatic identification of plant diseases with limited data. Plants 10 (1), 28.
Argüeso, D., Picon, A., Irusta, U., Medela, A., San-Emeterio, M.G., Bereciartua, A., Alvarez-Gila, A., 2020. Few-shot learning approach for plant disease classification using images taken in the field. Comput. Electron. Agric. 175, 105542.
Arnold, S.M., Mahajan, P., Datta, D., Bunner, I., Zarkias, K.S., 2020. learn2learn: a library for meta-learning research. arXiv e-prints. arXiv:2008.12284.
Barbedo, J.G.A., 2016. A review on the main challenges in automatic plant disease identification based on visible range images. Biosyst. Eng. 144, 52–60.
Barbedo, J.G., 2018. Factors influencing the use of deep learning for plant disease recognition. Biosyst. Eng. 172, 84–91.
Barbedo, J.G.A., 2019. Plant disease identification from individual lesions and spots using deep learning. Biosyst. Eng. 180, 96–107.
Bock, C., Poole, G., Parker, P., Gottwald, T., 2010. Plant disease severity estimated visually, by digital photography and image analysis, and by hyperspectral imaging. Crit. Rev. Plant Sci. 29 (2), 59–107.
Boulent, J., Foucher, S., Théau, J., St-Charles, P.-L., 2019. Convolutional neural networks for the automatic identification of plant diseases. Front. Plant Sci. 10, 941.
Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C.F., Huang, J.-B., 2020. A closer look at few-shot classification. arXiv e-prints. arXiv:1904.04232.
Cunningham, P., Delany, S.J., 2020. k-nearest neighbour classifiers. arXiv e-prints. arXiv:2004.04523.
Elhassouny, A., Smarandache, F., 2019. Smart mobile application to recognize tomato leaf diseases using convolutional neural networks. 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), pp. 1–4.
Esgario, J.G., de Castro, P.B., Tassis, L.M., Krohling, R.A., 2022. An app to assist farmers in the identification of diseases and pests of coffee leaves using deep learning. Information Processing in Agriculture 9 (1), 38–47.
Esgario, J.G., Krohling, R.A., Ventura, J.A., 2020. Deep learning for classification and severity estimation of coffee leaf biotic stress. Comput. Electron. Agric. 169, 105162.
Geetharamani, G., Pandian, J.A., 2019. Identification of plant leaf diseases using a nine-layer deep convolutional neural network. Comput. Electr. Eng. 76, 323–338.
Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning. MIT Press Available at http://www.deeplearningbook.org.
Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E., 2020. Array programming with NumPy. Nature 585 (7825), 357–362.
He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778.
Hospedales, T., Antoniou, A., Micaelli, P., Storkey, A., 2020. Meta-learning in neural networks: a survey. arXiv e-prints. arXiv:2004.05439.
Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q., 2017. Densely connected convolutional networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4700–4708.
Hunter, J.D., 2007. Matplotlib: A 2d graphics environment. Comp. Sci. & Eng. 9 (3), 90–95.
Jadon, S., 2020. SSM-net for plants disease identification in low data regime. IEEE/ITU International Conference on Artificial Intelligence for Good (AI4G), pp. 158–163.
Koch, G., Zemel, R., Salakhutdinov, R., 2015. Siamese Neural Networks for One-Shot Image Recognition. ICML Deep Learning Workshop, p. 2.
Krohling, R., Esgario, J.G., Ventura, J.A., 2019. BRACOL - a Brazilian arabica coffee leaf images dataset to identification and quantification of coffee diseases and pests. Mendeley Data.
LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D., 1989. Backpropagation applied to handwritten zip code recognition. Neural Comput. 1 (4), 541–551.
Li, F.-F., Krishna, R., Xu, D., 2021. Cs231n: convolutional neural networks for visual recognition lecture notes. Stanford University https://cs231n.github.io/.
Manso, G.L., Knidel, H., Krohling, R.A., Ventura, J.A., 2019. A smartphone application to detection and classification of coffee leaf miner and coffee leaf rust. arXiv e-prints arXiv:1904.00742.
Mohanty, S., Hughes, D., Salathe, M., 2016. Using deep learning for image-based plant disease detection. Front. Plant Sci. vol. 04, 7.
Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. Pytorch: an imperative style, high-performance deep learning library. Adv. Neural Inf. Proces. Syst. 32, 8024–8035.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.
Rahman, C.R., Arko, P.S., Ali, M.E., Khan, M.A.I., Wasif, A., Jani, M.R., Kabir, M.S., 2018. Identification and recognition of rice diseases and pests using deep convolutional neural networks. arXiv e-prints arXiv:1812.01043.
Ravi, S., Larochelle, H., 2017. Optimization as a model for few-shot learning. International Conference on Learning Representations (ICLR).
Ruder, S., 2016. An overview of gradient descent optimization algorithms. arXiv e-prints. arXiv:1609.04747.
Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520.
Schroff, F., Kalenichenko, D., Philbin, J., 2015. FaceNet: a unified embedding for face recognition and clustering. arXiv e-prints arXiv:1503.03832.
Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv e-prints arXiv:1409.1556.
Snell, J., Swersky, K., Zemel, R., 2017. Prototypical networks for few-shot learning. Advances in Neural Information Processing Systems, pp. 4077–4087.
Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P.H., Hospedales, T.M., 2018. Learning to compare: Relation network for few-shot learning. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1199–1208.
Suzuki, N., Rivero, R.M., Shulaev, V., Blumwald, E., Mittler, R., 2014. Abiotic and biotic stress combinations. New Phytol. 203 (1), 32–43.
Tan, M., Le, Q., 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. International Conference on Machine Learning, pp. 6105–6114.
Tassis, L.M., Tozzi de Souza, J.E., Krohling, R.A., 2021. A deep learning approach combining instance and semantic segmentation to identify diseases and pests of coffee leaves from in-field images. Comput. Electron. Agric. 186, 106191.
Van der Maaten, L., Hinton, G., 2008. Visualizing data using t-SNE. J. Mach. Learn. Res. 9 (86), 2579–2605.
Ventura, J.A., Costa, H., de Santana, E.N., Martins, M.V.V., 2017. Manejo das doenças do cafeeiro conilon. Café Conilon 435–479.
Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al., 2016. Matching networks for one shot learning. Advances in Neural Information Processing Systems, pp. 3630–3638.
Wang, B., Wang, D., 2019. Plant leaves classification: a few-shot learning method based on siamese network. IEEE Access 7, 151754–151763.
Wang, Y., Yao, Q., Kwok, J.T., Ni, L.M., 2020. Generalizing from a few examples: a survey on few-shot learning. ACM Comput. Surv. 53 (3).
Waskom, M.L., 2021. seaborn: statistical data visualization. J. Open Source Softw. 6 (60), 3021.