Comparison of ResNet and EfficientNet CNN Models for Plant Disease and Pest Detection

Artur Correia
DETI, UA
Universidade de Aveiro
Aveiro, Portugal
NMEC: 102477

Daniel Carvalho
DETI, UA
Universidade de Aveiro
Aveiro, Portugal
NMEC: 77036

***Abstract*—The goal of this project was the application of different Convolutional Neural Network (CNN) architectures for the resolution of the plant disease and pest detection problem, based on the examples present in the PlantVillage dataset. For this, different ResNet and EfficientNet models were trained and compared based on various performance metrics, with these models being either trained fully from scratch on the previously mentioned dataset, or trained with transfer learning from other models pre-trained on either the broad ImageNet dataset, or a more specific plant disease dataset curated by other researchers in previous work. The models trained from scratch proved to be more efficient, with ResNet-9 achieving a validation accuracy of 99.37% and a validation loss of 0.020, while the best performing pre-trained model had a validation accuracy of 97.98% and validation loss of 0.096. EfficientNet managed to compete with ResNet, while offering much lower training times, making this architecture a very good solution for real-world applications.**

***Index Terms*—Machine Learning, Classification, PlantVillage, Supervised Learning, CNN, ResNet, EfficientNet, Plant Disease Detection, Hyper-Parameter Tuning**

I. Introduction

A. The Importance of Pest and Disease Detection on Crops

One of the major problems facing the human civilization in the decades to come is related to the increasing overpopulation, and the difficulty our society has finding and adapting the resources it needs to sustain this growth, with estimates by the UN suggesting that the population will reach the 9 billion people mark well ahead of 2050. [1] The biggest hardship when it comes to supporting this number of people will be the nutritional needs of such a large number of people, with FAO, the Food and Agriculture Organization of the UN, predicting that the rate of food growth has to increase at least 70% by 2050, with a total crop production of 9.4 billion tons. [2]

However, food and crop security is constantly threatened by a big number of factors, such as the decline of pollinators or the appearance of plant diseases and pests, which are further enhanced by the increasing adverse effects of climate change. [3] In fact, plant diseases are estimated to account for 20%-40% of the global decline in food production. [4] This is particularly egregious due to the fact that a big portion of the food produced in the world is done so by smallholder farmers, that is, farmlands managed by families and consisting of less than two hectares, with estimates by FAO suggesting that a third of the world's crops are produced by these farmlands, and with their share of agricultural land being even higher in the developing world, in regions such as South Asia and Sub-Saharan Africa. [5] These farms are typically poorer, not having a lot of resources to deal with such threats, and with the production of crops ensuring the livelihood of the farmers, with their destruction by diseases and pests thus risking not only the world food supply chain, but also the living conditions and standards of the crop producers.

In order to try to prevent and combat these problems, a wide use of pesticides has been researched and applied, with this being one of the factors in the increased food production since the 1950s. [6] However, these substances have known environmental hazards, causing negative impacts on biodiversity and the biological populations of the environments where they are applied, as well as decreasing soil, air and water quality, and even constituting a risk for human health. [7] This further increases the impact of climate change, constituting a self-fulfilling cycle.

Thus, the paradigm is changing towards the implementation of technologies and techniques that allow for the fast and cheap detection of diseases and pests affecting crops, with quick detection being essential for the correct management of agricultural production, as it enables early treatment and increases crop yields. [4] This early detection is possible due to the distinct traces of lesions and symptoms these diseases typically show on plant's leaves, flowers and fruits. [4]

Traditionally, this identification has been supported by agricultural extension organization and other institutions, particularly in the cases of poorer regions where the previously mentioned smallholder farmers prevail, with experts being trained to manually identify the traces of the diseases. However, this is a very laborious, time-consuming and costly endeavor. [4] Thus, the focus has been shifted towards the application of tools that take advantage of the rapid uptake and availability of mobile phone and internet technology worldwide, with the increased quality of mobile phone cameras, as well as the high performance processes of phones, allowing to apply diseases diagnosis based on automated image recognition in an unprecedented scale, and in a much cost-efficient way for farmers. [3]

Thus, the use of computer vision and the application of

deep learning techniques to solve this problem has seen a major uptick in research in recent years, due to the importance of the its resolution, but also the myriad of potential commercial solutions that would profit from the development of technologies employing these models for crops' disease and pest detection.

B. Machine Learning Algorithms

Initial research for the application of machine learning algorithms to solve the crop disease detection problem applied traditional models for image classification, that used conventional image processing algorithms, or manual design of features and classifiers, to extract the features from images, such as color, texture and shape to train the used classifiers. These models included the SIFT and SURF algorithms. [3] Even though they had success in detecting some diseases such as leaf blotch and rust, they have limitations when it comes to accurately identifying subtler symptoms of diseases, and early-stage disease detection, while also struggling to process complex and high-resolution images. [8]

However, these methods have been upstaged by the appearance of deep-learning techniques, such as deep belief networks (DBNs) and convolutional neural networks (CNNs). In these methods, the use of deep-layered neural networks allows for the automatic learning and extraction of relevant and hidden features and patterns present in large amounts of data. [8]

The convolutional layers of CNNs, in particular, can be seen as matching filters that are derived directly from data, producing an hierarchy of visual representations that are optimized for the specific task at hand, and having a large capacity for generalization, processing data never observed before and enabling robustness to background heterogeneity. [6]

Thus, when compared with the previous machine learning models, the use of CNN allows the reduction of the time spent in applying feature engineering and image enhancement tecnhiques, while also allowing for the testing of more classes simultaneously, outdoing these algorithms on all performance benchmarks. [3] Though they yield more accurate results, they have the disadvantage of requiring higher processing powers and large sets for training. They are still, by far, more cost effective than the traditional methods used for plant detection described in the previous section, while reducing the subjective error underlying these methods. [4]

To aid in the development and training of these models, public datasets are created and made available to the broad public. One example of such datasets is the PlantVillage dataset, an open access repository of imagens curated by Salathe et al. in 2016 that encompasses over 50.000 images of healthy and infected leaves of crop plants of various species and varieties, precisely with the aim of training potential machine learning solutions for the crop disease detection problem. [8] The dataset used on the current project is a dataset that further expanded the PlantVillage dataset, and publically available on Kaggle. This dataset will be further described on Chapter 2. Meanwhile, on the next section, an exploratory and brief review of related projects where this dataset was used will be made.

C. Related Work

The first work were the CNN model was applied for the plant disease detection problem was published by Sladojevic et al. in 2016, using an adapted version of the CaffeNet model and reaching an accuracy estimation of around 96% for the detection of 13 diseases made by a dataset curated by the researchers. [9]

Meanwhile, the PlantVillage dataset was first used on a publication by Mohanty et al., also in 2016, in which the GoogLeNet and AlexNet architectures were applied. In this project, the researchers used models trained from scratch from the PlantVillage dataset, and models that were developed using transfer learning based on the ImageNet dataset, with results showing better accuracies for the models where transfer learning was used. [3]

In fact, transfer learning was found to be quite ubiquitous in research projects done on this image detection problem using CNN, with the ImageNet dataset being the most widely used. This dataset contains millions of hand-annotated images, of more than 20000 categories, and has been instrumental in advancing computer vision and deep learning research, with various new models and architectures of CNN appearing in contests promoted by this project, such as the previously mentioned AlexNet. [10]

The reason why transfer learning is so widely used is because the first layers of the CNNs usually learn more generic low-level features that are not class-specific, allowing for greater generalization of the models. Thus, pre-trained models can be used when the amount of training data is relatively limited, as is often the case in the context of the plant disease detection datasets, while allowing for saving in terms of processing time and capacity. [6]

In more recent years, the ResNet architecture has proved to become the most common model in the research papers found on this field, obtaining the best accuracies and performances, due to its deeper network layers that managed to overcome the degradation problem. [11]

For instance, Brahimi et al. in 2018 tested the ResNet34 architecture on the PlantVillage dataset, accomplishing accuracy averages of around 95% for their custom models, as well as accuracies of 98% for models pre-trained on the ImageNet dataset. [12] Fuentes et al. in 2022 tested the ResNet-50 and ResNet-101 architectures as features extractors in object detection problems, in a curated dataset based on field photographs of plants. [12]

Other researchers tested custom made CNN models, but usually obtaining smaller accuracy values when compared with ResNet studies, with Shobana et al. [13] getting an accuracy of 95% or Kolli et al. getting an accuracy of 94%. [14] Besides the work already mention in this section, two research projects in particular garnered our interest and inspired the work done by us in this project.

The first, published by Dong et al. in 2023, [4] explores the effects that using pre-trained models on generic datasets, like the case of the previously discussed ImageNet, could have when applied to a more specific problem like the plant disease detection. Dong et al. argue that using these models might lead to weak generalization and precision, because of the lack of domain knowledge of plant diseases, in particular, the lack of sufficient knowledge of plant phenotypes, leading to poor accuracy in classification, detection and segmentation models. [4] Thus, they aimed to create pre-trained plant models specifically trained on the problem domain to be used instead of these general pre-trained models, for transfer learning purposes, and based on a compiled dataset of over 400000 images containing 40 plant species and 120 plant diseases classes. To assess the performance of their model, they then trained three ResNet 101 architecture: one using transfer learning from the ImageNet dataset; one using transfer learning from the pre-trained model created from their curated plant disease dataset; and one trained from scratch on their test dataset. The custom trained model had worse results, of around 85% accuracy, whereas the pre-trained model from the ImageNet dataset had 90% average accuracy, and the pre-trained model from their compiled dataset had a 95% validation accuracy. Their pre-trained models are publicly available and will be used in our work. [4]

The second project catching our attention was from Kaggle, where the dataset was taken from. In a notebook published there by Ingle in 2019, an adapted version of ResNet 9 was used, obtaining test accuracies of 99% in the dataset used on this project. [15] As part of the processing of the model training, a few optimizations not widely discussed in the research papers found were made to prevent overfitting of the data, which include learning rate scheduling, the addition of a regularization technique by weight decay, and gradient clipping. These methods will be adapted in the ResNet architectures applied on the current work, and thus will be further described on Chapter X.

During our research, we also came across another CNN architecture, EfficientNet, created by Tan et al. in 2019, which uniformly scales all dimensions of depth, width and resolution using a compound coefficient. [16] This architecture has been highly praised and has presented very strong performance metrics on the projects that have applied it so far, being named one of the "most powerful CNN architectures" that currently exist. [16], [17] We haven't however, seen it yet applied on research papers using the PlantVillage dataset, with this becoming one of our aims for this work.

D. *Project Goals*

The main goals for the current work were the comparison of the performance of different ResNet and EfficientNet architectures on the Plant Village test dataset, using either transfer learning from pre-trained models on the ImageNet dataset, transfer learning from the pre-trained models on the compiled plant dataset created by Dong et al., and models trained from scratch and custom made by us.

With this, we aim to corroborate the established efficiency of the different ResNet model on the plant disease classification problem, checking how this architecture is influenced by the different methods of transfer learning, the deepness of the various architecture available, and checking whether the use of the optimization techniques described by Ingle can improve its results compared with the existing work. Besides, we also aim to compare the EfficientNet's performance to the ResNet models, to see if this model is well adjusted for the plant disease classification or not.

II. Exploratory Data Set Analysis and Data Pre-Processing

A. *Data Set Description*

The "New Plant Diseases Dataset," initially created by Samir Bhattarai, is accessible on Kaggle and expands upon the Plant Village dataset. The Plant Village dataset comprises over 50,000 images of healthy and unhealthy plant leaves, collected from experimental research stations associated with Land Grant Universities in the USA. The technicians working at these stations gathered leaves from the plants under study and placed them against a paper sheet, which served as a grey or black background. The images were captured outdoors in full light, intentionally varying between strong sun and cloudy conditions to simulate a range of real-world scenarios. The photographs were taken using a standard point-and-shoot camera, specifically the Sony DSC-Rx100/13 with 20.2 megapixels, in automatic mode. In some cases, when the leaf size was too large, such as with corn leaves, multiple images were taken of different sections of the leaf and later edited and merged. [18]

To enhance the Plant Village dataset, offline augmentation techniques were employed, involving various transformations and modifications applied to the dataset before training. While the dataset does not explicitly specify the transformations performed, a closer examination suggests that rotation, cropping, and scaling were the most likely augmentations applied. [19]

The dataset comprises approximately 87,000 RGB images with dimensions of 256 x 256 pixels, depicting both healthy and unhealthy crop leaves. These images are categorized into the same 38 different classes as the original dataset, as shown in Figure 1.

B. *Data Analysis*

Further analysis of the dataset reveals the presence of 14 distinct plant species, each associated with various diseases. In total, 26 different diseases have been identified across the plant species.

Additionally, an examination of the distribution of images per class, as illustrated in Figure 2, indicates a well-balanced dataset. Each class contains around 1,800 to 2,000 images, which suggests that there is no need for any preprocessing steps. The images have already been prepared for training, ensuring they have the correct size, angle, and cropping. Furthermore, the balanced distribution of images across classes

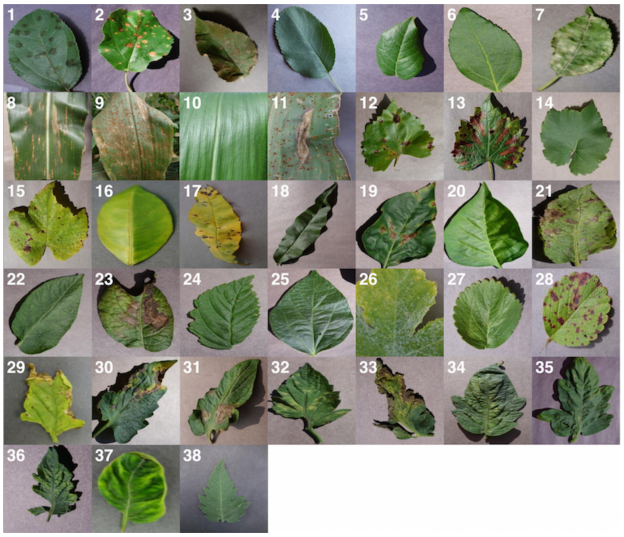


Fig. 1: (1) AppleScab, Venturiainaequalis (2) Apple BlackRot, Botryosphaeriaobtusa (3) AppleCedarRust, Gymnosporangium juniperi-virginianae (4) Applehealthy (5) Blueberryhealthy (6) Cherryhealthy (7) Cherry PowderyMildew, Podoshaeraclandestine (8) CornGrayLeafSpot, Cercosporazeae-maydis (9) CornCommonRust, Pucciniasorghi (10) Cornhealthy (11) Corn NorthernLeafBlight, Exserohilumturcicum (12) GrapeBlackRot, Guignardiabidwellii, (13) GrapeBlackMeasles(Esca), Phaeomoniellaaleophilum,Phaeomoniella chlamydospora (14) GrapeHealthy (15) GrapeLeafBlight, Pseudocercosporavitis (16) OrangeHuanglongbing(CitrusGreening), CandidatusLiberibacterspp. (17) PeachBacterialSpot, Xanthomonascampestris (18) Peachhealthy (19) BellPepperBacterialSpot, Xanthomonascampestris (20) BellPepperhealthy (21) Potato EarlyBlight, Alternariasolani (22) Potatohealthy (23) PotatoLateBlight, Phytophthorainfestans (24) Raspberryhealthy (25) Soybeanhealthy (26) SquashPowdery Mildew, Erysiphecichoracearum (27) StrawberryHealthy (28) StrawberryLeafScorch, Diplocarponearlianum (29) TomatoBacterialSpot, Xanthomonascampestris pv.vesicatoria (30) TomatoEarlyBlight, Alternariasolani (31) TomatoLateBlight, Phytophthorainfestans (32) TomatoLeafMold, Passalorafulva (33) TomatoSeptoria LeafSpot, Septorialycopersici (34) TomatoTwoSpottedSpiderMite, Tetranychusurticae (35) TomatoTargetSpot, Corynesporacassiicola (36) TomatoMosaicVirus (37) TomatoYellowLeafCurlVirus (38) Tomatohealthy.

further supports the suitability of the dataset for training purposes.

The next step in the process was to load the data, and we chose to utilize PyTorch, an open-source machine learning framework known for its efficient tensor computations. It uses tensors as the fundamental data structure, which are like multi-dimensional arrays or matrices. One of the key advantages of the framework is its ability to move tensors to the GPU, enabling faster computations. This was a significant factor in our decision to work with PyTorch, as it allowed us to train our models efficiently by leveraging GPU acceleration. [20]

Since the dataset had already been divided into an 80/20 ratio for training and test data, we opted to maintain this partitioning structure for our project. This decision was based on its suitability for our specific requirements and allowed us to maintain consistency with related studies.

It is worth noting that we chose not to utilize grayscale versions of the images in our project. This decision was based on previous related work, which indicated that RGB images performed significantly better than grayscale images. [3]

III. Implemented Machine-Learning Models

A. *CNN*

A Convolutional Neural Network (CNN) is a powerful deep learning algorithm specifically designed for image recognition and processing tasks. It consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. [21]

Convolutional layers are the core components of CNNs. They apply filters to input images, extracting features like edges, textures, and shapes. These filters, also known as kernels, slide over the input image, performing element-wise multiplications and summations to produce feature maps. These feature maps capture important patterns and characteristics of the input images. [21]

Pooling layers are used to down-sample the feature maps, reducing their spatial dimensions while preserving essential information. Pooling operations, such as max pooling or average pooling, consolidate the most relevant features by selecting the maximum or average values within specific regions. This helps to reduce the computational complexity and the number of parameters in the subsequent layers. [21]

The output of the pooling layers is fed into one or more fully connected layers. These layers have neurons connected to all the neurons in the previous layer. They process the extracted features and make predictions or classifications based on the learned patterns. The fully connected layers are responsible for capturing high-level features and generating the final output of the CNN. [21]

CNNs are trained using large datasets of labeled images. During the training process, the network learns to recognize patterns and features that are associated with specific objects or classes. By adjusting the weights and biases of the network through backpropagation, the CNN learns to minimize the difference between predicted outputs and ground truth labels. [21]

Once trained, a CNN can be used to classify new images by passing them through the network and obtaining predictions. Additionally, the learned features from the intermediate layers of CNNs can be extracted and utilized in other applications, such as object detection or image segmentation.

The vanishing gradient problem is a challenge that can occur during the training of deep neural networks, including convolutional neural networks (CNNs). It arises when the gradients computed during backpropagation become extremely

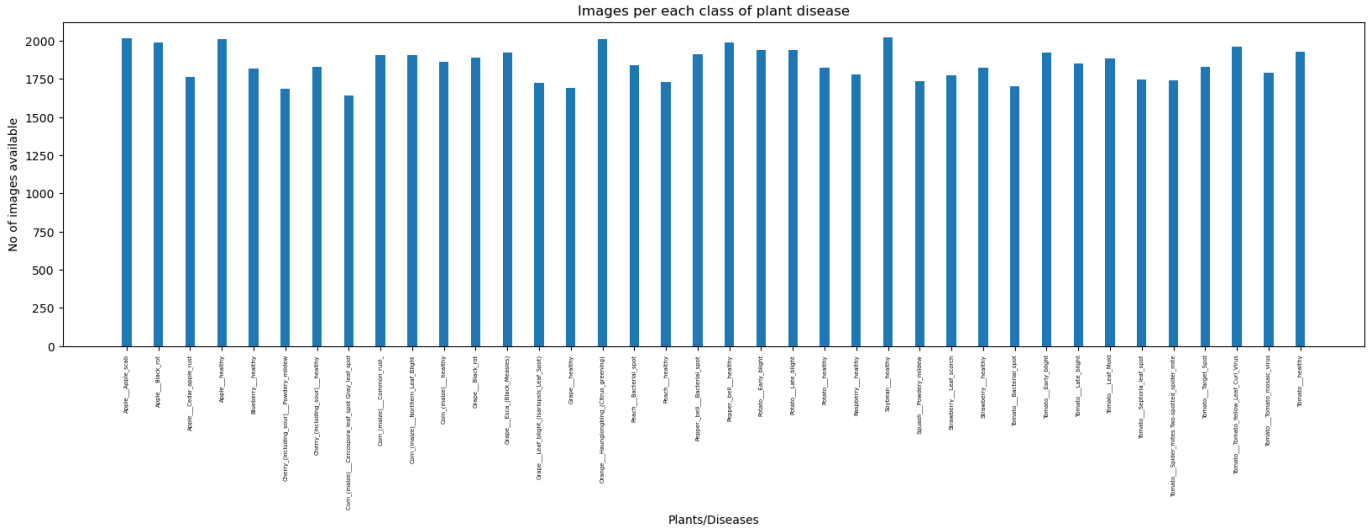


Fig. 2: Class Distribution

small as they propagate through multiple layers of the network. As a result, the weights in the earlier layers receive negligible updates, leading to slow or ineffective learning. [22]

In traditional deep networks, as the gradients are backpropagated from the output layer to the input layer, they tend to shrink exponentially due to the chain rule of differentiation and the use of activation functions with limited ranges, such as the sigmoid or hyperbolic tangent function. Consequently, the updates to the weights in the earlier layers become increasingly insignificant, impeding the learning process. [22]

Solutions to this problem include Batch Normalization, gradient clipping, and faster hardware. However, one of the newest and most effective solutions is the use of residual networks, or ResNets, which have been extensively explored in this project.

B. ResNet

In 2015, researchers at Microsoft Research introduced Residual Networks (ResNets), a groundbreaking architecture that revolutionized image recognition and processing tasks. ResNets were designed to address challenges encountered during the training of deep neural networks, such as vanishing or exploding gradients. This architecture introduced the concept of residual blocks and skip connections, which enabled the successful training of very deep networks. [23]

ResNets tackled the problem of vanishing or exploding gradients by incorporating residual blocks. Instead of expecting layers to directly learn the underlying mapping, ResNets allowed the network to fit the residual mapping. This was accomplished through skip connections that connected the activations of a layer to subsequent layers, bypassing some intermediate layers. By doing so, ResNets facilitated the learning of the residual mapping, expressed as H(x) = F(x) + x, where F(x) represents the desired underlying mapping, and x is the input to the residual block. [23]

Skip connections offered several advantages in ResNets. Firstly, they provided a form of regularization by allowing the network to bypass any layer that may hinder overall performance. This enabled the training of very deep neural networks without suffering from degraded performance due to vanishing or exploding gradients. Secondly, skip connections simplified the learning process by focusing the network's attention on learning the residual function F(x) rather than the entire mapping H(x). [23]

ResNets were constructed by stacking multiple residual blocks together, each consisting of convolutional layers and skip connections. Through this stacking, ResNets effectively learned hierarchical representations of increasing complexity. The depth of the network, ranging from 100 to 1000 layers, was determined by the number of stacked residual blocks. [23]

Another architecture utilizing skip connections is Highway Networks, which, like LSTM, employed parametric gates to control the information flow through the skip connection. However, despite the inclusion of skip connections, Highway Networks have not achieved superior accuracy compared to ResNets. [23]

To validate the effectiveness of ResNets, the authors of the original paper conducted experiments on the CIFAR-10 dataset. They trained ResNets with depths ranging from 100 to 1000 layers and obtained remarkable accuracy, surpassing that of traditional deep neural networks. The skip connections and residual blocks in ResNets played a crucial role in enabling the successful training of such deep architectures. [23]

C. EfficientNet

EfficientNet is a CNN model proposed by Tan et al. of Google in 2019, designed to achieve state of the art performance, while maintaining efficiency in terms of computational resources, that has seen some wide adoption in research over the past few years. [16]

The aim of this work was to allow for easier scaling of neural networks that allowed for the provision of better accuracy while increasing the network size, by uniformly scaling all dimensions of depth, width and resolution using grid-search, to produce a simple but highly efficient compound scaling coefficient that allows for better performance and efficiency, with this project leading to the creation of a new architecture represented by EfficientNet. [16] In this context, depth refers to the number of layers of the network, width refers to the number of convolution kernels, and resolution refers to the input images dimension. [16]

After its creation, the authors tested the dataset on many transfer learning datasets, like CIFAR-100 or ImageNet, achieving state of the art accuracy and efficient, when compared with previous CNN models.

The efficiency and effectiveness of the EfficientNet model allows its deployment on a wide range of devices, which includes devices with limited computational power, such as mobile devices or embedded systems. [16] This is possible because of some key features from the architecture:

- The compound scaling coefficient previously described, that allows for all the dimensions of the network to be balanced.
- The adoption of mobile inverted residual blocks, which was a key component of the prior MobileNet V2 model, specifically created to be mobile-friendly.This block consists of depthwise convolution, followed by pointwise convolution, and a skip connection, reducing the computational cost of the model, due to being more memory efficient as a results of the removal of non-linearities in the narrow layers. [24]
- The introduction of a lightweight attention mechanism called “squeeze and excitation”, that recalibrates channel-wise features.

Various sub models of EfficientNet exist. The B0 model is the base one, and the smallest and most computationally efficient, being the one explored in this project. Figure 3 presents the baseline architecture of EfficientNet B0. [24]

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Fig. 3: EfficientNet Architecture

IV. Model Training Experimental Setup

The main objective of this project is to compare ResNet models of different sizes, focusing on their accuracy and efficiency for the classification task at hand. As mentioned earlier, ResNet models vary in the number of residual blocks they contain, and we aim to determine which ResNet architecture performs optimally in our classification task by evaluating these different sizes.

Another aspect we intend to examine is the utilization of pre-trained models using other datasets, such as the Plant dataset and ImageNet dataset mentioned earlier, in addition to training the models solely on the main dataset of this project. By incorporating pre-trained models, we can assess the impact of transfer learning on the performance and efficiency of the ResNet models.

Moreover, we are interested in exploring the potential of using EfficientNet as an alternative model for our classification task. EfficientNet is renowned for its exceptional performance and efficiency, achieved through a compound scaling method that strikes a balance between model size and accuracy. By comparing EfficientNet with the ResNet models, we can gain insights into whether EfficientNet is a suitable choice for our project in terms of accuracy and efficiency.

A. *Model Architecture*

The selection of ResNet sizes for this project was based on a thorough review of related work, the availability of pre-trained models for transfer learning, and the computational resources at our disposal.

We considered multiple sources of pre-trained models, including the Plant Phenomics study done by Dong et al.(2023) [4], which provided ResNet 34 and ResNet 50 architectures specifically designed for image-based plant disease diagnosis.

Additionally, we referred to the Kaggle notebook by Atharva Ingle titled ”Plant Disease Classification - ResNet-99.2%” [15], which presented a custom ResNet 9 architecture. This architecture was incorporated into our project to expand the range of ResNet architectures considered.

To enable comparisons with the Plant Phenomics architectures, we utilized pre-trained ResNet models from PyTorch trained on the ImageNet dataset. This allowed us to include ResNet 34 and ResNet 50 architectures. We also included ResNet 18 to compare with a custom architecture discussed in the subsequent section.

Due to limitations in computational resources, we were unable to train deeper versions of ResNet models, such as the ResNet 100 mentioned in the Plant Phenomics study. Even training ResNet 50 was only possible with the use of pre-trained models.

As the primary objective of our work was to compare different architectures using various pre-trained and fully trained models, we made the decision to exclude the deeper ResNet versions.

In addition to ResNet models, we incorporated the EfficientNet_B0 architecture from PyTorch as an alternative. This architecture is pre-trained on the ImageNet dataset and was employed for both transfer learning and fully training approaches. Although we experimented with EfficientNet_B1, we found that its performance did not significantly differ from EfficientNet_B0. Considering the increased complexity and

computational requirements of larger EfficientNet models, we chose to focus solely on EfficientNet_B0 for this project.

Table I gives a summary of every model employed, being the custom architectures fully analysed in the next section.

TABLE I: Models used and datasets

Model	Fully Trained	Plant Phenomics (Pre-Trained)	ImageNet (Pre-Trained)
ResNet-9	x		
ResNet-18	x	x	x
ResNet-34	x	x	x
ResNet-50		x	x
EfficientNet_B0	x		x

To accommodate our specific requirements, we developed custom architectures based on the ResNet base implementation found in the documentation and a GitHub repository by the user kuangliu [A9]. These custom architectures consisted of BasicBlocks and Bottleneck blocks.

The BasicBlock represented the fundamental residual block in the ResNet architecture. It consisted of two 3x3 convolutional layers with batch normalization, followed by a shortcut connection. The shortcut connection was implemented using a 1x1 convolutional layer and batch normalization to adjust the dimensions of the input if necessary. The forward method applied ReLU activation after each convolutional layer and added the shortcut connection to the output.

The Bottleneck represented a specialized residual block used in deeper ResNet architectures to reduce computational complexity. It comprised three convolutional layers, with filter sizes of 1x1, 3x3, and 1x1, respectively. Batch normalization was applied after each convolutional layer. Like the BasicBlock, the Bottleneck block included a shortcut connection to adjust the input dimensions if required. The forward method applied ReLU activation after each convolutional layer and added the shortcut connection to the output.

By combining these blocks, we created the ResNet model. The model was initialized with the number of blocks for each layer and the number of classes for classification. The architecture included a 3x3 convolutional layer, batch normalization, and four layers, each consisting of a specified number of blocks. After the last layer, average pooling was applied, followed by flattening the output. Finally, a linear layer was used for classification.

As mentioned earlier, the EfficientNet model was utilized from the PyTorch framework, being the default model in the documentation.

B. Training of the models

Before commencing the detailed training of the models, it is crucial to familiarize ourselves with the hardware setup. We had access to two PCs, with one being significantly faster than the other due to its more powerful GPU. However, through testing, we concluded that the GPU's power did not impact the accuracy of the models; rather, it mainly influenced the training time. Table II shows the specifications of the PCs used.

TABLE II: Environment Settings during training

Particular	**Description**	
	PC1	**PC2**
Operating system	Endeavouros Cassini Nova 03-2023 R2	Windows 11 Home 22H2
GPU	NVIDIA GeForce GTX 1650 Ti	NVIDIA GeForce RTX 3060
CPU	Intel(R) Core(TM) i7-10750H	AMD Ryzen 7 5800H

After loading and transforming the images into tensors using the PyTorch framework, we utilized the CUDA parallel computing platform and API to transfer the tensors to the GPU. This enabled us to leverage the GPU's computational power and accelerate the training process significantly. To optimize memory usage, the images were divided into batches and sent to the GPU for processing. Additionally, we ensured that the batches were shuffled before being sent to introduce randomness during training.

For pre-trained models, an additional step was taken to freeze all layers except the last classification layer. This involved modifying the last linear layer to output the correct number of classes for our specific classification task. By freezing the other layers, their weights remained unchanged while only the last classification layer was trained.

The training process was adapted from a Kaggle notebook that had already incorporated several effective techniques. These included:

- **Learning rate scheduling**: Instead of using a fixed learning rate, we implemented a learning rate scheduling technique known as the "One Cycle Learning Rate Policy." This involved starting with a low learning rate and gradually increasing it batch-by-batch until reaching a high learning rate for 30% of the epochs. Subsequently, the learning rate gradually decreased to a very low value for the remaining epochs.
- **Weight decay**: To prevent the weights from becoming too large during training, we applied weight decay as a regularization technique. Weight decay involves adding an additional term to the loss function, which helps control the magnitude of the weights.
- **Gradient clipping**: To ensure stable parameter updates and prevent undesirable changes caused by large gradient values, we implemented gradient clipping. This technique limits the gradient values to a specified range, promoting more controlled updates to the model parameters.

Thus, the initial training parameters included the number of epochs, the maximum learning rate for the learning rate scheduling, the gradient clipping threshold, the weight decay coefficient, and the choice of optimizer function. By default, we used the Adam optimizer for all models, with the values for the other parameters in initial training being the ones used on

this notebook. In the subsequent sections, we will delve into the hyperparameter tuning made for some of these parameters in more detail.

All the trained models were saved for future use. By saving the models, we preserved their learned parameters and architecture, allowing us to load and utilize them later without the need for retraining.

C. *Hyperparameter Tuning*

After evaluating the performance of the custom fully trained models (that is, the models in which we didn't use transfer learning) we weren't fully content with the observed results, which are further explored in the next Chapter.

Thus, we decided to do hyper-parameter tuning to try and increase their efficiency and accuracy. The strategy used for the training of these custom models, adapted from GitHub repositories and the Kaggle notebook we explored and described in the previous section, allowed for the definition of the following hyperparameters:

- Number of epochs
- Batch size
- Maximum learning rate
- Gradient clipping
- Weight decay
- Optimization function

Though ideally, we would like to train various possible values for all of these parameters, using a technique such as GridSearch with K-fold cross validation, this wasn't possible, mainly because the searching of the hyperparameter space took a lot of training time and required very strong computational power, that we didn't have access to while developing this project.

Thus, we decided to train a more limited range of values for the two hyperparameters we considered striked the best balance between usefulness and potential for optimization of the models against the time and processing power required for the training of the models. These were the maximum learning rate, that can significantly affect the convergence and generalization of the model, and which we varied between 0.001 and 0.1, and the gradient clipping, which can help control overfitting by regularization of the model's weights and which we varied between 0.0001 and 0.001. Tuning was made for all the custom ResNet and EfficientNet models.

In the end, the optimized parameters allowed for a much better of all these models, as is further expanded on the next Chapter.

D. *Performance Evaluation Metrics*

To compare the performance of the various optimized ML models, while testing with the test set, the models were evaluated based on some performance metrics. Most of these metrics were based on the obtained confusion matrixes, and are listed below:

- **Accuracy**, the number of correctly classified instances divided by the total number of instances.
- **Recall**, the true positive rate.
- **Precision**, the fraction of correctly classified positive samples from all the samples classified as positive.
- **F1-Score**, which determines the weighted average of Precision and Recall
- **Balanced Accuracy**, which calculates the average of the per-class accuracy, giving equal weight to each class, no matter whether the set is balanced or not.

Given that the dataset was well balanced, as previously described, we considered the use of validation accuracy as the main metric to compare the different trained models.

Besides these metrics, the models were also compared based on their loss rates during training:

- **Training loss**, which compares the value of the loss function computed on the train dataset during each iteration of the training process. This loss represents how well the model is fitting the training data.
- **Validation loss**, which compares the values of the loss function computed on a separate validation dataset that the model has not seen during training, calculated after each epoch of the training. This metric provides an estimate of how well the model generalizes to unseen data.

V. Results and Analysis

In this section, we will present and analyze the performance of the various ResNet and EfficientNet architectures we trained with the Plant Village dataset. The aim of the following discussion is the comparison of the efficiency and accuracy of the different models, but also determing the effects that transfer learning from models either pre-trained with the ImageNet dataset or the plant dataset curated by Dong et al. (2023) [4] had when developing the models, when compared with the models trained only in the dataset. We will then compare the results with the conclusions obtained from the previously presented related work. To do so, we will use the performance methods described on the previous chapter.

A. *Transfer Learning From Models Pre-Trained with the ImageNet Dataset*

As previously mentioned, three different ResNet models with different sizes, as well as an EfficientNet model, were trained using transfer learning from models trained with the ImageNet dataset.

Table III presents the validation accuracy, train loss and validation loss, balanced accuracy and training time of each model, as well as the weighted averages for the precision, recall and F1-score obtained for each of the test classes. The matrixes corresponding to these values are presented in Appendix B. Furthermore, the graphs detailing the accuracy and loss evolution with the number of epochs, as well as the learning rate evolution through the batches, are also presented in the same appendix.

As can be seen, the validation accuracy values for the models are all very satisfying, ranging from the 96% to the 98% mark. The same happens with the remaining performance metrics. When it comes to the ResNet models, these values

TABLE III: Transfer Learning From Models Pre-Trained With ImageNet Dataset Performance Metrics

Model	Validation Acc	Train Loss	Validation Loss	Weighted Averages			Balanced Acc	Training Time
				Precision	Recall	F1-Score		
ResNet-18	96.23%	0.211	0.129	96%	96%	96%	96.20%	3'19"
ResNet-34	95.76%	0.228	0.145	96%	96%	96%	95.73%	5'39"
ResNet-50	97.98%	0.096	0.078	98%	98%	98%	98.00%	8'22"
EfficientNet B0	97.01%	0.212	0.098	99%	96%	98%	96.16%	8'10"

slightly increase in the deeper 50 model, which was the overall best performing model in this group. The EfficientNet model manages to be competitive with this model, while having similar training times.

When it comes to the train and validation loss, the values are also satisfying, in particular when considering that this is a relatively complex classification problem, involving a big number of possible classes. With the training loss, this means that the model is fitting the training data quite well, with the value decreasing a lot during training, as was expected. Meanwhile, the validation loss is consistently quite lower than the training loss, suggesting that the models can generalize well to unseen data, with overfitting having been, thus, prevented, with no further adjustments to regularization methods being required. Both the train loss and the validation losses follow the trend of the remaining metrics, being lower with deeper ResNet models, and with the EfficientNet model being competitive with the best performing ResNet-50 model.

When analyzing the extended classification report for each class, two trends were detected. Firstly, the classes corresponding to the healthy examples of plants have, overall, higher values in the performance metrics of all the trained models, reaching quite frequently the 100% mark in all of the evaluated metrics. Secondly, the examples related with all the classes involving tomatoes, both healthy and with disease (like bacterial spot, mosaic virus or blight), are noticeably lower than with previous classes – while previous classes have at worst values of 96 to 97% for these metrics, the tomato classes see values as low as 90% in the best performing ResNet-50 architecture, for the recall in the tomato with target spot disease class.

B. Transfer Learning From Models Pre-Trained with the Plant Dataset

In this section, the results obtained for the models trained with transfer learning from the models pre-trained with a curated plant dataset by Dong et al. are presented. As was the case with the previous models, Table IV shows the main performance metrics results for this training. Appendix B presents the confusion matrixes and shows the graphs detailing the accuracy and loss evolution with the number of epochs, as well as the learning rate evolution through the batches.

As explained on the previous chapter, we could only train two ResNet models using this transfer learning technique, as they were the only ResNet pre-trained models made available by the research team. The validation accuracy, as well as the values for the other performance metrics, are quite satisfactory and comparable to the values obtained for the ImageNet pre-trained models (with the validation accuracy slightly decreasing from the 96-98% range to the 96-97% range). It's also verified that these values slightly increase with the deeper 50 architeture.

However, the training and validation losses stay the same in both models, with values quite larger than what was observed for the ImageNet pre-trained models – for the ImageNet models, the 34 and 50 ResNet architectures had training losses of 0.228 and 0.096 and validation losses of 0.145 and 0.078 respectively, while for the models presented in this section these values were 0.228 and 0.310 for the training loss, and 0.145 for the validation loss. This might indicate that, surprisingly, the pre-trained model on the curated plant dataset might have a harder time generalizing to the examples present in the Plant Village dataset. This could be due to the fact that the curated dataset has a much small size compared to the vast ImageNet dataset, increasing these model's tendencies for overfitting. It could also be due to class imbalance problems present in the curated plant dataset created for the pre-training of these models, as Dong et al. stated that they used datasets with coverages focusing only on certain species, such as apples and cassavas, that might have lead for the appearance of some bias in the models. In fact, while some classes had a number of examples higher than 10000, other classes had only around 100 images in the curated dataset. [4] Furthermore, many of the classes they used are not present in the test dataset used for this project. It's also possible that these models would require more fine-tuning to reach their complete potential in this project, for example in the number of epochs, as the specificity of these models for the plant detection problem could come at the cost of higher initial losses during the training process. However, further fine-tuning wasn't possible due to the lack of available computational power.

The observations described on the ImageNet models regarding the higher performance metrics for healthy classes when compared with disease classes, and the overall lower performance metrics for the tomato classes, were also verified in these models, with some metrics reaching values even lower than observed in the ImageNet models for the best performing ResNet 50 models (the lowest metric on the models analyzed in this section was of 85% precision score for the tomatoes with target spot disease, versus 90% for the ImageNet models).

C. Fully Trained Models

The last group of models trained for this project where the modules where no transfer learning was used – instead, these models were trained from scratch, using the Plant

TABLE IV: Transfer Learning From Models Pre-Trained With Dong et al.'s Custom Dataset Performance Metrics

Model	Validation Acc	Train Loss	Validation Loss	Weighted Averages			Balanced Acc	Training Time
				Precision	Recall	F1-Score		
ResNet-34	95.76%	0.228	0.145	96%	96%	96%	95.73%	5'41"
ResNet-50	96.79%	0.310	0.145	97%	97%	97%	96.77%	8'28"

Village dataset. As previously stated, we initially trained the various ResNet and EfficientNet models on this group using the hyperparameters found in the work done by the Kaggle notebook that inspired our training method for these custom architectures. Afterwards, hyperparameter tuning was made, and the models were retrained with the optimal parameters.

Table V presented the performance metrics for the initial training of these models.

The first obvious observation is that the training times for these models increase considerably, when compared with the training times for the pre-trained models. Plus, the model with the best performance was by far the ResNet-9 architecture, achieving the highest validation accuracy of the project so far (98.36%), while getting the smallest validation loss (0.050). The deeper 18 and 34 ResNet architectures had increasingly worse metrics, with 34 achieving only a validation accuracy of 92.55% and reaching a validation loss as high as 0.366. Though EfficientNet wasn't able to compete with ResNet 9, it still had better metrics than the remaining 2 models, in particular when it comes to validation loss, while achieving a very efficient training time of under half an hour, comparable with the smallest ResNet 9 architecture.

Given that this bucked the trend observed for the pre-trained models, and that the accuracies are quite lower than any of the other models trained on the project so far, this observation was what prompted us to do hyperparameter tuning, to try and see if we could optimize this models, and to check whether the ResNet-9 architecture would still be the best performing one in the experiment.

As previously mentioned, we only did hyperparameter tuning on the learning rate and weight decay parameters. We wished to do the tuning of more hyperparameters, with more varied possible values under study, but that would require the application of a grid search technique that wasn't viable with the computational and time limits the project was under. The results for the optimal hyperparameter values obtained were of 0.001 learning rate and 1e-4 weight decay for every model, except for EfficientNet, where a 0.0001 learning rate was optimal.

Meanwhile, Table VI presents the performance metrics for all the models after hyperparameter tuning. As was the case with previous sections, Appendix B has the respective confusion matrixes, and the accuracy, loss and learning rate variation graphs.

It's immediately apparent that all the models highly increased their performance after hyperparameter tuning. For the ResNet architectures, the validation accuracy of the ResNet-18 and 34 models increased to values of 97%, while the validation loss decreased to values of 0.09. Meanwhile, ResNet9 increased to great values of 99.37% accuracy, while only holding a 0.020 validation loss.

Meanwhile, EfficientNet managed to reach a value of 98% validation accuracy, and only 0.072 validation loss, while having very high values of around 98% for all other metrics. This puts it on par with the best ResNet model (ResNet-9), while being much more efficient when it comes to training time, taking only 11 minutes to train, and thus, proving the worthiness of studying EfficientNet models for the solution of the plant disease classification problem.

As was the case with the previous models, it was once again verified that healthy classes have higher performance metrics, while tomato classes overall have lower performance scores. In the case of the best performing ResNet-9 model, healthy classes regularly reached 100% for all metrics. Tomato classes had values of 92-100%, whereas other classes had values of at least 96%. Once again, the target spot afflicted tomatoes class was the worst performing of the group.

D. Analysis of Results

When looking at the set of performance metrics obtained for all of the trained models in the project, it becomes quite apparent that all models performed in a satisfying manner: the values of accuracy ranged from 95.76% (obtained for ResNet-34 in both the pre-trained models with ImageNet and the custom plant dataset) to 99.37% (obtained in the fully trained ResNet-9 model); the validation loss ranged from as low as 0.020 (in the same ResNet-9 model) to 0.145 (in both models pre-trained with the custom plant dataset); and the remaing performance metrics ranged from the lowest value of 96% to 99%. Thus, all models, from the ResNet architecture to the EfficientNet architecture, and from the models using transfer learning to the fully trained models, are quite promising and could be further researched and improved to be used in real life applications.

When comparing the models trained by transfer learning, no major performance differences are noticed between the models pre-trained with the ImageNet dataset and the models pre-trained with the custom dataset curated by Dong et al, [4] whose results are found on Table III and Table IV , respectively. In fact, the validation accuracies are in the range of 96-97% for all models, while the validation loss is in a similar range of 0.078-0.145. When looking at the ResNet models in particular, it was found that these performed better with deeper architectures.

Initially, we expected the models with transfer learning from the curated plant dataset to perform better, as the images it contains are more specific and personalized to the problem domain explored in this project. However, a few reasons

TABLE V: Performance Metrics of the Models Trained From Scratch Before Hyperparameter Tuning

Model	Validation Acc	Train Loss	Validation Loss	Weighted Averages			Balanced Acc	Training Time
				Precision	Recall	F1-Score		
ResNet-9	98.36%	0.214	0.050	98%	98%	98%	98.34%	26'33"
ResNet-18	93.44%	0.498	0.221	94%	93%	93%	93.42%	4h6'41"
ResNet-34	92.55%	0.516	0.366	93%	93%	92%	92.53%	3h25'39"
EfficientNet B0	96.37%	0.349	0.114	96%	96%	96%	96.32%	24'28"

TABLE VI: Performance Metrics of the Models Trained From Scratch After Hyperparameter Tuning

Model	Validation Acc	Train Loss	Validation Loss	Weighted Averages			Balanced Acc	Training Time
				Precision	Recall	F1-Score		
ResNet-9	99.37%	0.113	0.020	99%	99%	99%	99.24%	29'56"
ResNet-18	97.38%	0.328	0.096	96%	96%	96%	96.09%	3h25'28"
ResNet-34	97.27%	0.369	0.091	97%	97%	97%	97.26%	2h53'35"
EfficientNet B0	97.71%	0.219	0.072	98%	98%	98%	97.70%	11'42"

previously discussed could be the causes for this model not doing as well as expected: this might be because the curated dataset has a much smaller size when compared to the vast ImageNet dataset (the former has around 1-2 million examples, while the latter has 14+ million examples), which could have increased the pre-trained models chance of overfitting to that specific dataset, and thus not performing as well against the PlantVillage dataset used in the current work. It could also be because of the previously discusses class imbalance problems found in this curated dataset. However, with further fine-tuning of the training of the models used in this transfer learning technique, we expect their performance to be better, and eventually surpass the models with transfer learning from the ImageNet dataset.

The models trained from scratch using the Plant Village dataset, however, had by far the best results, with values of validation accuracy ranging from 97.27% (in ResNet-34) to 99.37% (in ResNet-9), and validation losses in the interval of 0.020 (in ResNet-9) to 0.096 (in ResNet-18). However, this was only the case after hyperparamer tuning was made – before this, the models had quite a significant lower performance metrics values, with the exception of the ResNet-9 model, whose performance already made it the best of the entire project even without the tuning. The values for the performance metrics in pre-tuning are presented in Table V, while the post-tuning results are in Table VI.

These results, in particular the lower validation loss values, suggest that these models are not only well fitted to the training Plant Village dataset, but are also the ones more suited to generalization, with overfitting to the training set having been avoided. As such, we can conclude that the optimization techniques implemented in the training of the models, described on the previous chapter, such as the learning rate scheduling, weight decay control and gradient clipping, did indeed work well as regularization techniques and in preventing problems of either vanishing or exploding gradients, allowing for high variance and overfitting to be avoided in the custom models. However, we did not expect the performance values to decrease with the deeper ResNet architectures – this could be because the dataset is relatively small, consisting of only around 80.000 examples, which could trouble the training of deeper neural networks, leading to suboptimal performances – in fact, with limited data, the optimization process of the deep networks could get trapped in suboptimal local values, preventing the model from converging to the best solution. It could also cause higher variance and overfitting of the values (which is visible by the small increase in the training and validation losses), leading to parameter sensitivity in the networks. It would have been interesting to test further deeper ResNet models to see if these observations were corroborated, in particular the 50 and 101 architectures. However, this was not possible due to a lack of computational and processing power in the computers used for the project.

When comparing the EfficientNet models with the ResNet models, it seems this architecture is able to compete with the best ResNet models – in the transfer learning with models pre-trained on the ImageNet dataset, the EfficientNet B0 model only scored worse than the ResNet-50 model, achieving a validation accuracy of 97.01% and validation loss of 0.098. Meanwhile, on the models trained from scratch, EfficientNet B0 only performed worse than ResNet-9, achieving a validation accuracy of 97.71% and validation loss of 0.072. However, the big advantage of this model is that it can obtain these good results in a fraction of the time it takes to train the ResNet models – for instance, the fully trained EfficientNet model took only 11 minutes to train, whereas the ResNet models ranged fom 30 minutes to 3 and a half hours. Thus, EfficientNet proves itself to be worthy of further research for application in this field, particularly because its architecture and very efficient training make it very well adjusted to be applied in systems with a lower computational power, such as mobile phones or embedded systems, which could precisely be the type of systems where models for plant disease and pest detection could be implemented in for crop control. When considering the training time in particular, it is also observed that even though the ResNet models had better performance metrics in the fully trained models when compared with their analogues in the transfer learning models, the fact that the difference wasn't significant and that the latter had much more efficient training times, due to the transfer learning process,

might mean that it's not worth it to fully train the ResNet models when using this approach in real systems.

Another final observation made when analyzing the results is that, across all trained models, the classes corresponding to the healthy examples of the crops in the dataset had better performance metrics when compared with the classes of the crops afflicted with diseases. This might be because there's a big variety in the format of lesions caused by the symptoms of the various diseases and pests that crops can develop, thus making it harder for the deep neural network models to detect patterns and correlations in the hidden features of these class, and thus, decreasing the efficiency of the model. It was also observed that all of the classes relating to tomatoes, whether healthy or afflicted by diseases, had worse performance metrics than the classes belonging to other crops. Thus, we recommend further studies to try and enrich the number of examples belonging to these classes, and maybe even training models based only on tomatoes datasets, to further increase the efficiency of disease detection on this crop. When comparing the results obtained in this project with those found in the related work explored earlier, this project corroborates the views that the ResNet architecture is very effective in the training of models for pest and diseases detection on crops. In fact, our models can compete with the best models found in related work, such as the 95% accuracy found by Brahimi et al. in their custom models [12], or the 95% accuracy found by Shobana et al., also on their own custom models. [13] Our work also corroborated the training optimization proposed by Ingle on their Kaggle project. [15]

We didn't, however, manage to corroborate Dong et al.'s find that their pre-trained models with their curated plant dataset outperform pre-trained models with the ImageNet dataset – though, as previously discussed, this could be because further fine-tuning of the training process was necessary.

VI. Conclusion

This project aimed to compare different ResNet and EfficientNet models applied in the plant disease and pest identification problem, and comparing the influence of having either models pre-trained with transfer learning from other datasets, or models fully trained from scratch on the PlantVillage dataset applied in the project.

As explored in the previous chapter, all models obtained very satisfactory results when evaluated through all of the performance metrics used in the project, with values of validation accuracy ranging from 95.76% (obtained for ResNet-34 in both the pre-trained models with ImageNet and the custom plant dataset) to 99.37% (obtained in the fully trained ResNet-9 model), and with very small validation losses, suggesting that the models are well adjusted and can be generalized to validate examples belonging to datasets different from the ones used for their training.

Though the results from the models trained via transfer learning were good, they didn't manage to outdo the models trained fully from scratch in the PlantVillage dataset. They did, however, manage to be much more time efficient, which could be an advantage in real-world applications. No difference was observed between pre-training with models from the general Image Net dataset, or models from the proposed plant specific dataset curated by Dong et al., though future work to improve the training of the latter models via fine-tuning could change these results.

EfficientNet proved to be a model with high potential for application in this problem, as it managed to compete with the best ResNet architectures in both the models trained with transfer learning and the models training from scratch, reaching validation accuracy values of 97.01% and 97.71% respectively, versus 97.98% and 99.37% for the best performing ResNet models. Their comparative much shorter training times might prove very advantageous in the deployment in mobile or embedded systems.

For future work, further exploration of both the EfficientNet and ResNet architectures is advised – further models from each should be tested and trained, such as the B7 model from the former, and the 101 model from the latter. We would also like to explore a more comprehensive set of values for hyperparameter tuning, as we were very limited by the time and processing power at our disposal – we could probably further optimize the models with the tuning of further hyperparameters.

The PlantVillage dataset should also be open to further updates in examples, as the current number makes it harder to fully explore the potential of the ResNet architecture, as the examples are not enough to properly train deeper models, which would theoretically yield better results.

The dataset should also be enriched with various examples of diseased crops, as the variety of lesions and symptoms observed mean that the performance of deep learning models in these classes are weaker. Ideally, a new dataset with the magnitude of the PlantVillage collection, but composed of photos of plants and crops leaves on the field, would be created and testes, as these are much closer to the conditions that would be found in real systems where these predictive models could be applied.

References

[1] UN Department of Economic and Social Affairs, Population Division. "World Population Prospects: The 2022 Revision." *United Nations*. Accessed June 22, 2023. https://www.un.org/en/global-issues/population#:~:text=The%20world%20population%20is%20projected,surrounding%20these%20latest%20population%20projections.

[2] World Food and Agriculture - Statistical Yearbook 2020. *World Food and Agriculture - Statistical Yearbook 2020*. (FAO, 2020). doi:10.4060/cb1329en.

[3] Mohanty, S. P., Hughes, D. P., Salathé, M. "Using deep learning for image-based plant disease detection." *Frontiers in Plant Science*, vol. 7, 2016.

[4] Dong, X. et al. "PDDD-PreTrain: A Series of Commonly Used Pre-Trained Models Support Image-Based Plant Disease Diagnosis." *Plant Phenomics*, vol. 5, 2023.

[5] FAO. "COVID-19 crisis threatens to double the number of people experiencing food insecurity." *Food and Agriculture Organization of the United Nations (FAO)*. Accessed June 22, 2023. https://www.fao.org/news/story/en/item/1395127/icode/

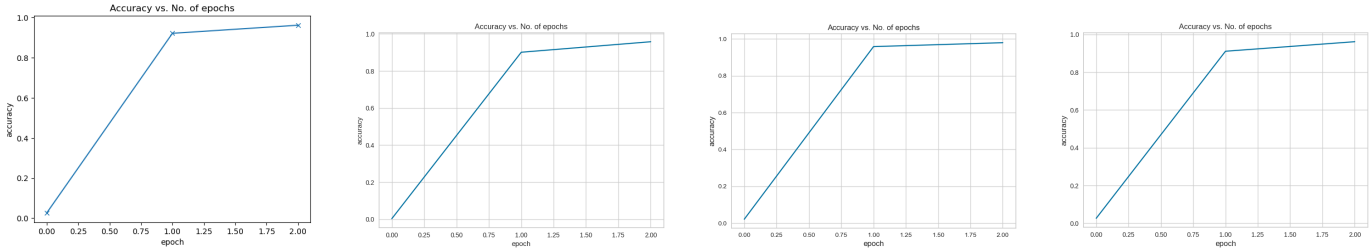
[6] Boulent, J., Foucher, S., Théau, J., St-Charles, P. L. "Convolutional Neural Networks for the Automatic Identification of Plant Diseases." *Frontiers in Plant Science*, vol. 10, 2019

[7] Government of British Columbia," Environmental Protection and Pesticides " 2017.
[8] Shoaib, M. et al. "An advanced deep learning models-based plant disease detection: A review of recent research." *Frontiers in Plant Science*, vol. 14, 2023.
[9] Sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., Stefanovic, D. "Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification." *Computational Intelligence and Neuroscience*, 2016.
[10] ImageNet. *ImageNet*. Accessed June 22, 2023. https://www.image-net.org/
[11] Mehta, P. "ResNet: The Most Popular Network in Computer Vision Era." *Towards Data Science*. Accessed June 22, 2023. https : / / towardsdatascience . com / resnet-the-most-popular-network-in-computer-vision-era-973df3e92809
[12] Brahimi, M. et al. "Deep Learning for Plant Diseases: Detection and Saliency Map Visualisation." Retrieved from https://www.researchgate.net/publication/325651831, 2018.
[13] Shobana, M. et al. "Plant Disease Detection Using Convolution Neural Network." In *2022 International Conference on Computer Communication and Informatics, ICCCI 2022*, 2022.
[14] Kolli, J., Vamsi, D. M., Manikandan, V. M. "Plant Disease Detection using Convolutional Neural Network." In *2021 IEEE Bombay Section Signature Conference, IBSSC 2021*, 2021.
[15] Atharva Ingle. "Plant Disease Classification using ResNet (99.2%)." *Kaggle*. Accessed June 22, 2023. https://www.kaggle.com/code/atharvaingle/plant-disease-classification-resnet-99-2#%F0%9F%8F%97%EF%B8%8F-Modelling-%F0%9F%8F%97%EF%B8%8F
[16] Tan, M., Le, Q. V. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," 2019.
[17] Jain, R. "Understanding EfficientNet: The Most Powerful CNN Architecture." *Medium*. Accessed June 22, 2023. https : / / medium . com / mlearning-ai / understanding-efficientnet-the-most-powerful-cnn-architecture-eaeb40386fad
[18] Hughes, D. P., Salathé, M. "An open access repository of images on plant health to enable the development of mobile disease diagnostics." Retrieved from http://www.fao.org/fileadmin/templates/wsfs/docs/expert_paper/How_to_Feed_the_World_in_2050.pdf.
[19] NVIDIA. "Offline Data Augmentation." *NVIDIA TAO Toolkit Documentation*. Accessed June 22, 2023. https://docs.nvidia.com/tao/tao-toolkit/text/offline_data_augmentation.html
[20] PyTorch. "PyTorch Tensors: A Deeper Tutorial." *PyTorch Tutorials*. Accessed June 22, 2023. https://pytorch.org/tutorials/beginner/introyt/tensors_deeper_tutorial.html
[21] GeeksforGeeks. "Convolutional Neural Network (CNN) in Machine Learning." Accessed June 22, 2023. https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/
[22] Wikipedia. "Vanishing Gradient Problem." *Wikipedia*. Accessed June 22, 2023. https://en.wikipedia.org/wiki/Vanishing_gradient_problem
[23] He, K., Zhang, X., Ren, S., Sun, J. "Deep Residual Learning for Image Recognition," 2015.
[24] Tan, M., Le, Q. V. "EfficientNet: Scaling of Convolutional Neural Networks Done Right." *Towards Data Science*. Accessed June 22, 2023. https : / / towardsdatascience . com / efficientnet-scaling-of-convolutional-neural-networks-done-right-3fde32aef8ff

VII. Appendix A: Project Contributions

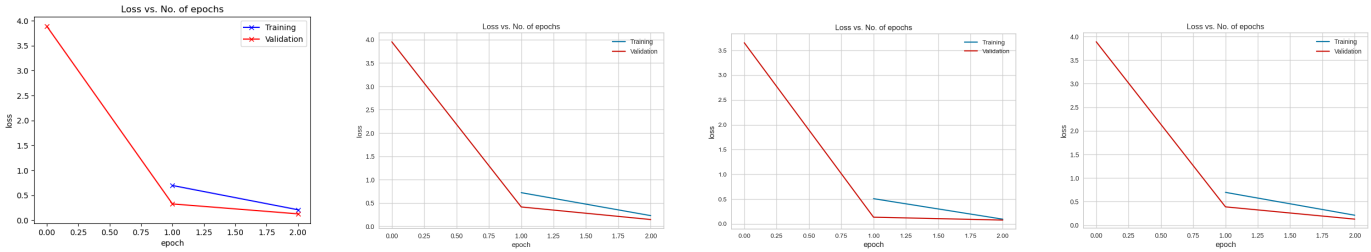
- Artur Correia - 50%
- Daniel Carvalho - 50%

VIII. Appendix B: Graphs and Confusion Matrices of the models trained



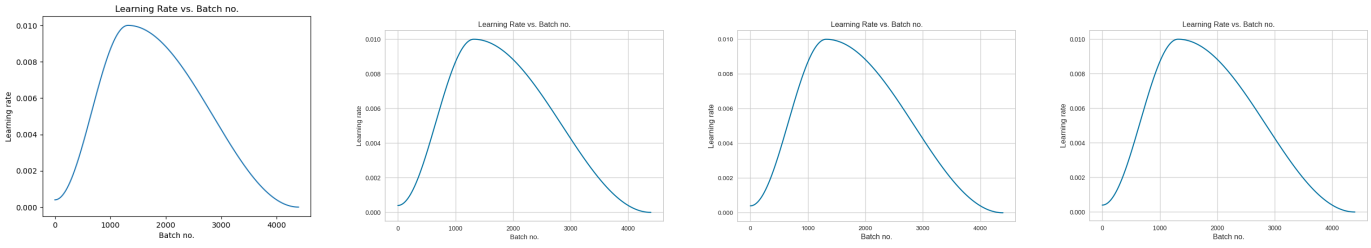
(a) ResNet-18 (b) ResNet-34 (c) ResNet-50 (d) EfficientNet_B0

Fig. 4: Pre-Trained models using ImageNet accuracy plots



(a) ResNet-18 (b) ResNet-34 (c) ResNet-50 (d) EfficientNet_B0

Fig. 5: Pre-Trained models using ImageNet loss plots



(a) ResNet-18 (b) ResNet-34 (c) ResNet-50 (d) EfficientNet_B0

Fig. 6: Pre-Trained models using ImageNet learning rate plots

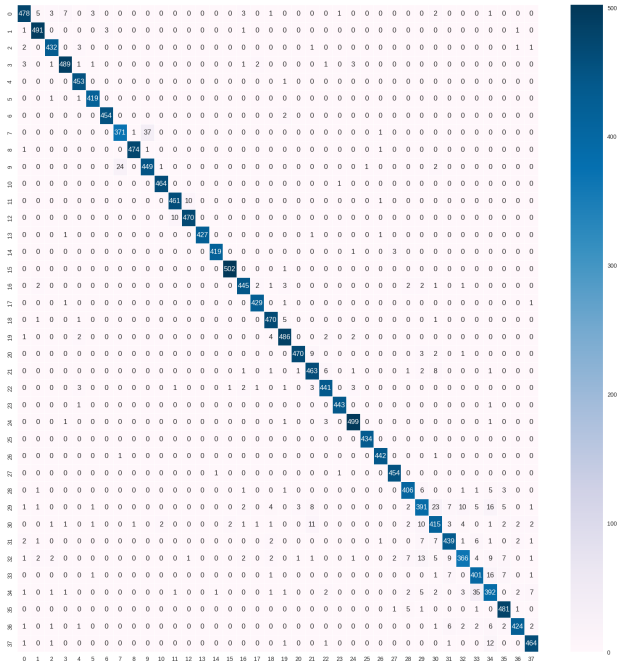


Fig. 7: ResNet18 pre-trained with ImageNet heatmap

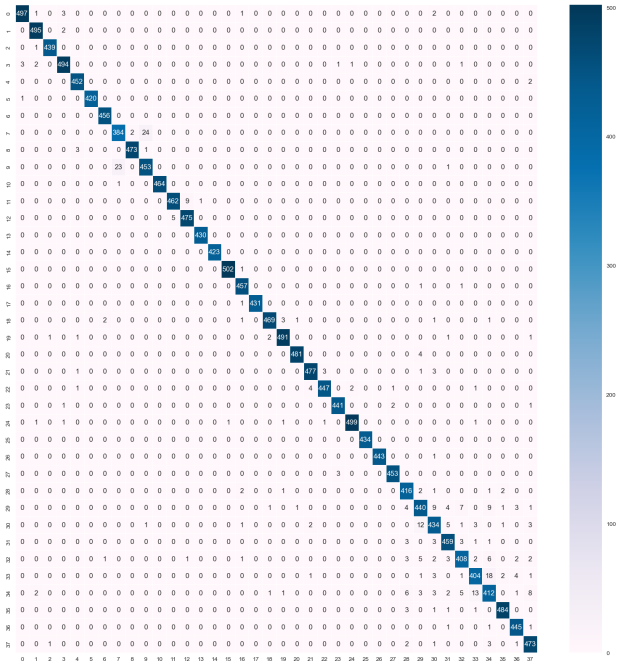


Fig. 9: ResNet50 pre-trained with ImageNet heatmap

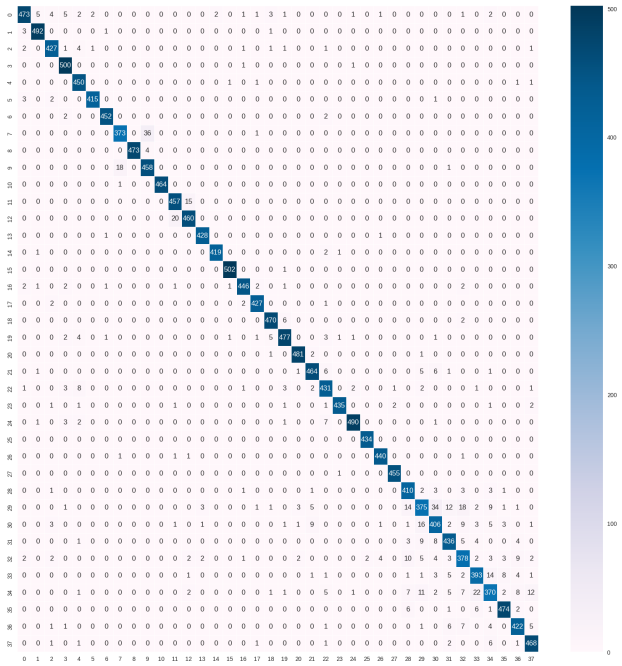


Fig. 8: ResNet34 pre-trained with ImageNet heatmap

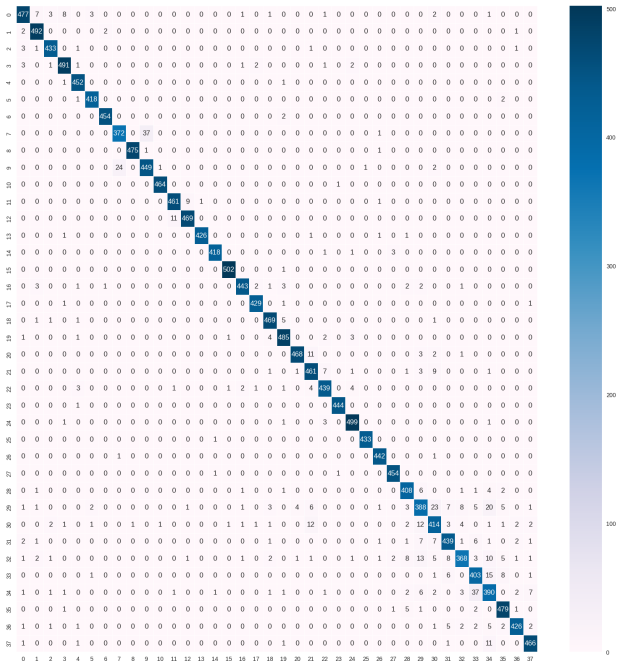
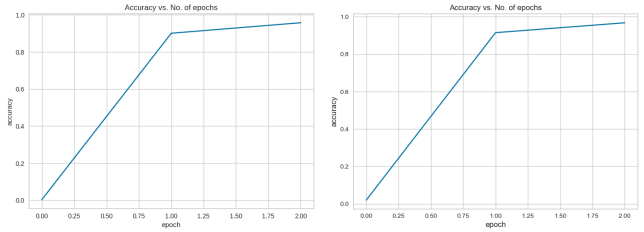
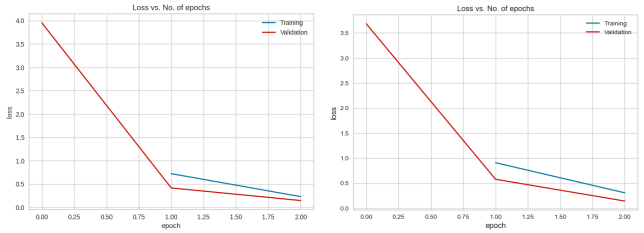


Fig. 10: EfficientNet_B0 pre-trained with ImageNet heatmap



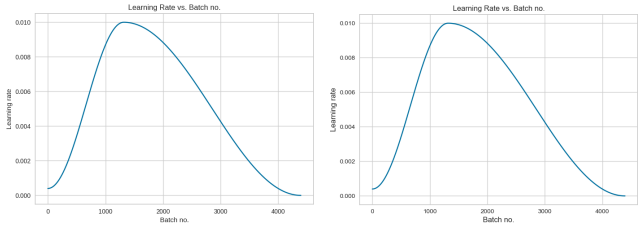
(a) ResNet-34 (b) ResNet-50

Fig. 11: Pre-Trained models using Plant Dataset accuracy plots



(a) ResNet-34 (b) ResNet-50

Fig. 12: Pre-Trained models using Plant Dataset loss plots



(a) ResNet-34 (b) ResNet-50

Fig. 13: Pre-Trained models using Plant Dataset learning rate plots

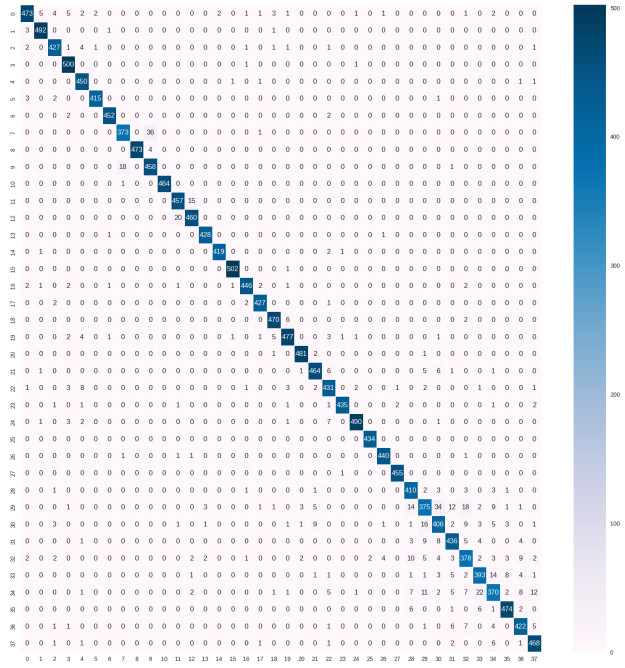


Fig. 14: ResNet34 pre-trained with Plant Dataset heatmap

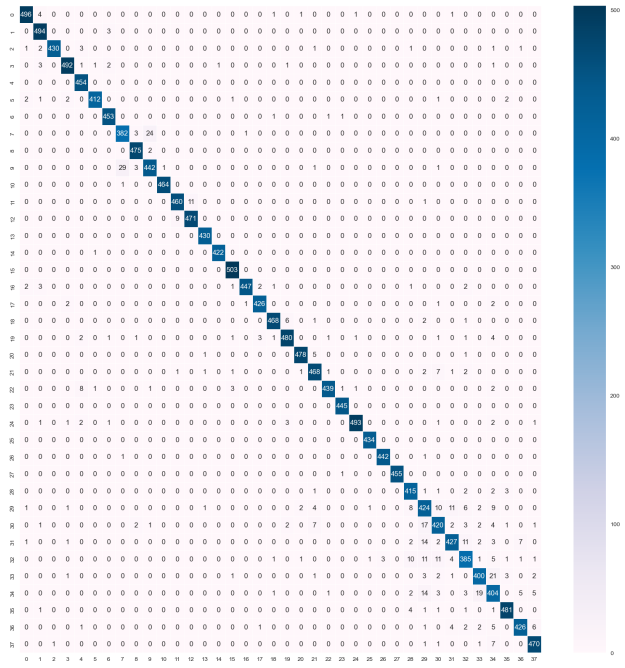
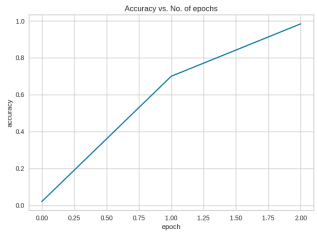
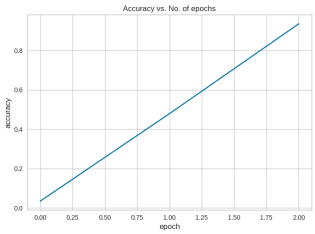


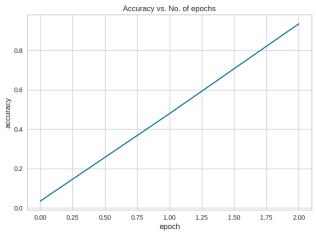
Fig. 15: ResNet50 pre-trained with Plant Dataset heatmap



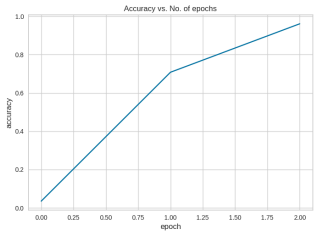
(a) ResNet-9



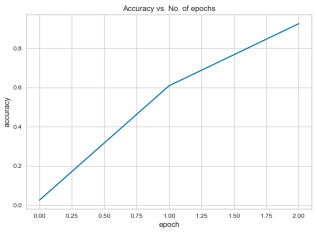
(b) ResNet-18



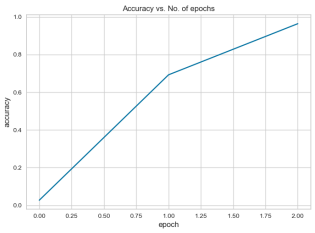
(c) ResNet-18 with tunning 1



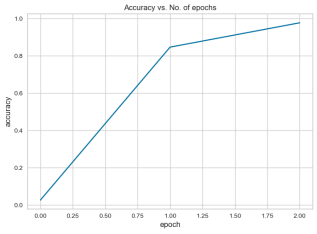
(d) ResNet-18 with tunning 2



(e) ResNet-34

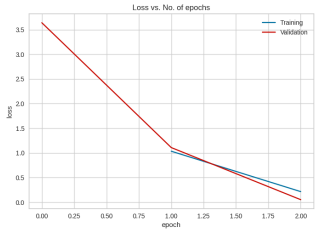


(f) EfficientNet_B0

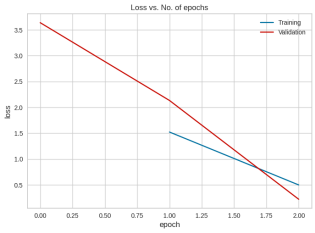


(g) EfficientNet_B0 with tunning

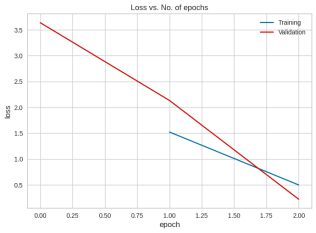
Fig. 16: Fully trained models accuracy plots



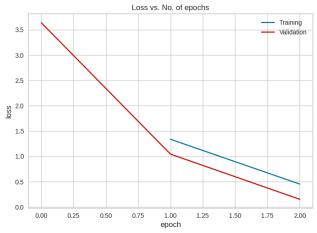
(a) ResNet-9



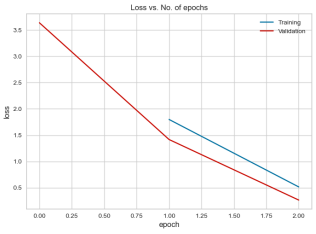
(b) ResNet-18



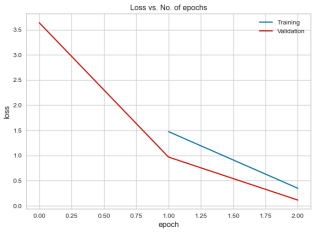
(c) ResNet-18 with tunning 1



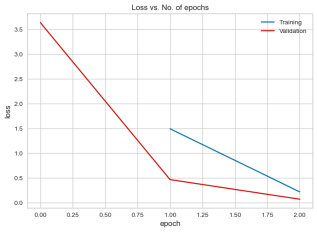
(d) ResNet-18 with tunning 2



(e) ResNet-34

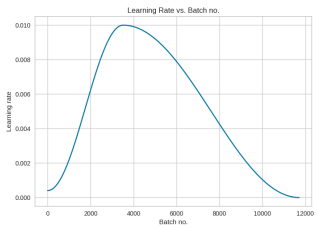


(f) EfficientNet_B0

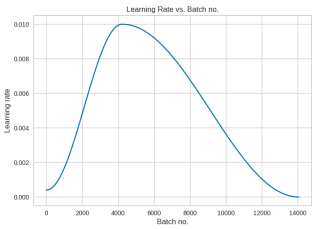


(g) EfficientNet_B0 with tunning

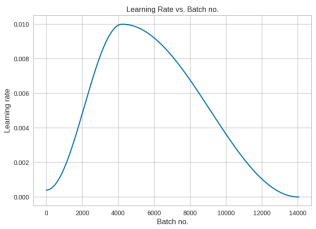
Fig. 17: Fully trained models loss plots



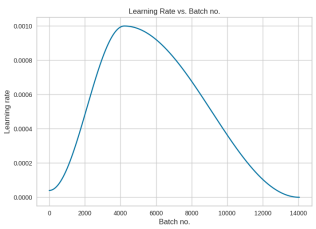
(a) ResNet-9



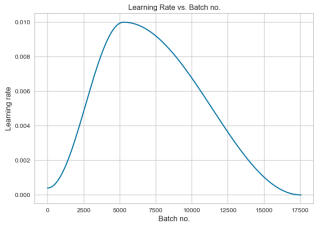
(b) ResNet-18



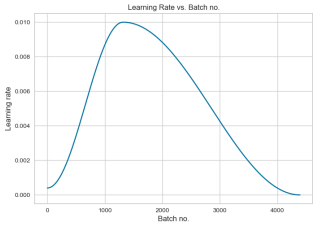
(c) ResNet-18 with tunning 1



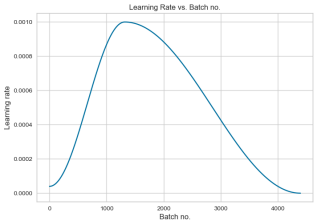
(d) ResNet-18 with tunning 2



(e) ResNet-34



(f) EfficientNet_B0



(g) EfficientNet_B0 with tunning

Fig. 18: Fully trained models learning rate plots

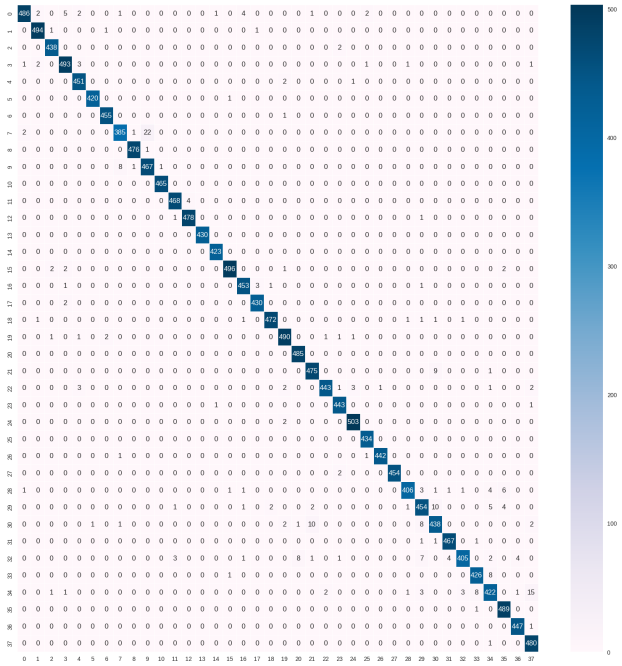


Fig. 19: Fully trained ResNet9 heatmap

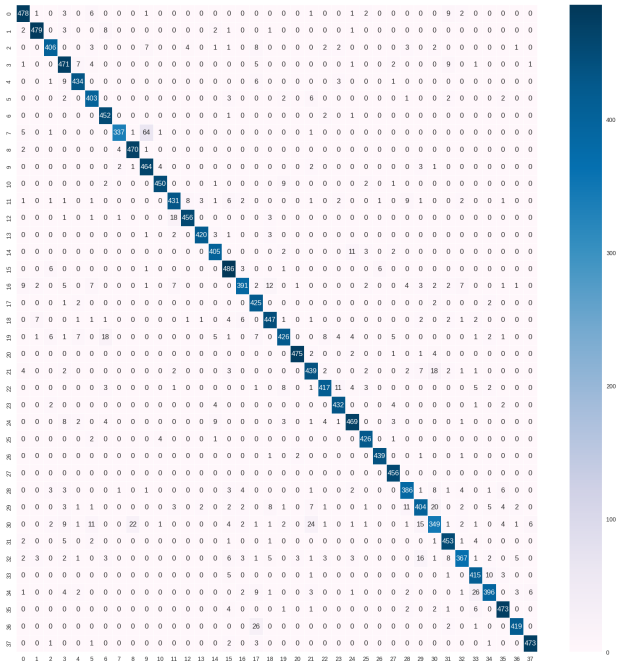


Fig. 21: Fully trained ResNet18 with tunning 1 heatmap

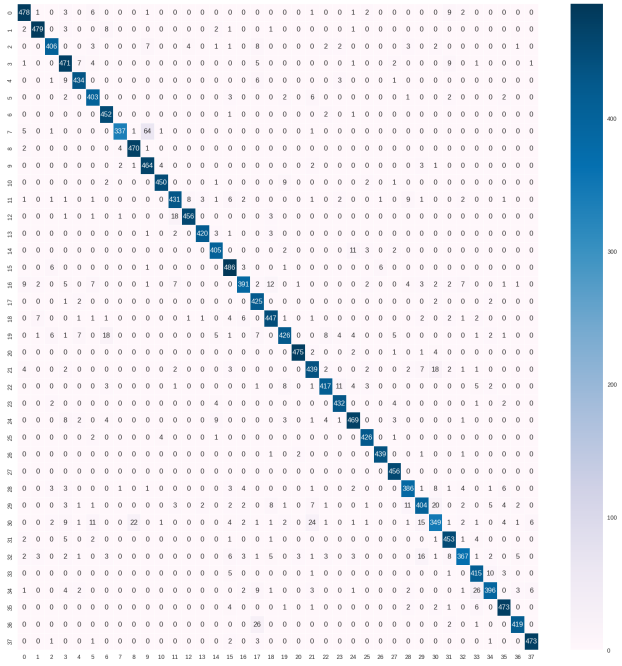


Fig. 20: Fully trained ResNet18 heatmap

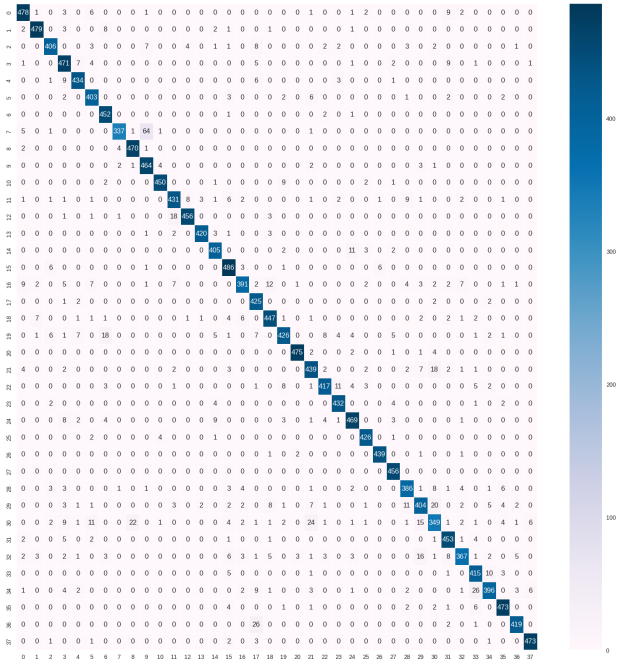


Fig. 22: Fully trained ResNet18 with tunning 2 heatmap

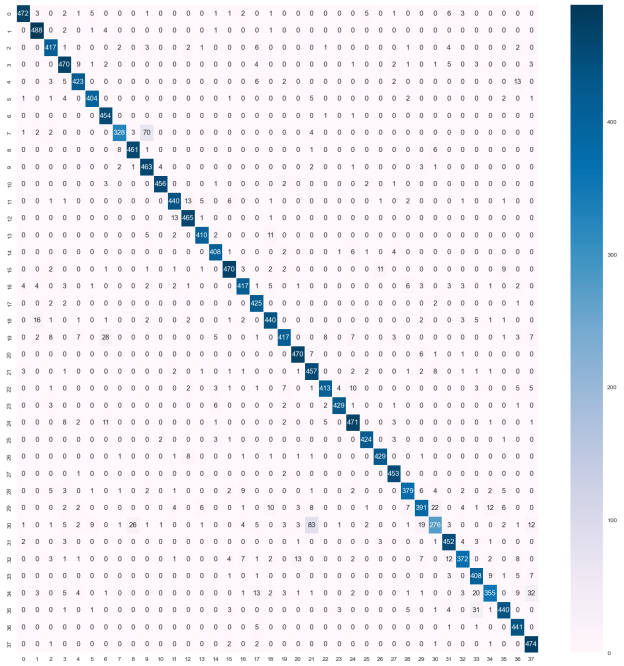


Fig. 23: Fully trained ResNet34 heatmap

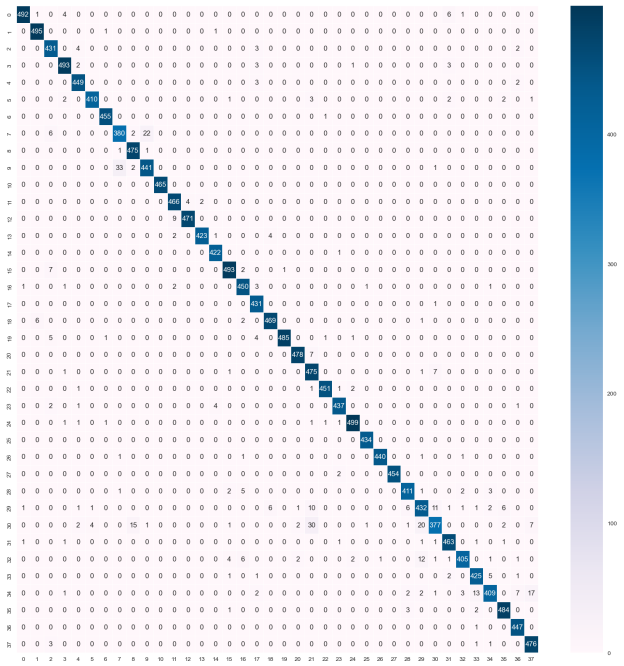


Fig. 24: Fully trained ResNet34 with tunning heatmap

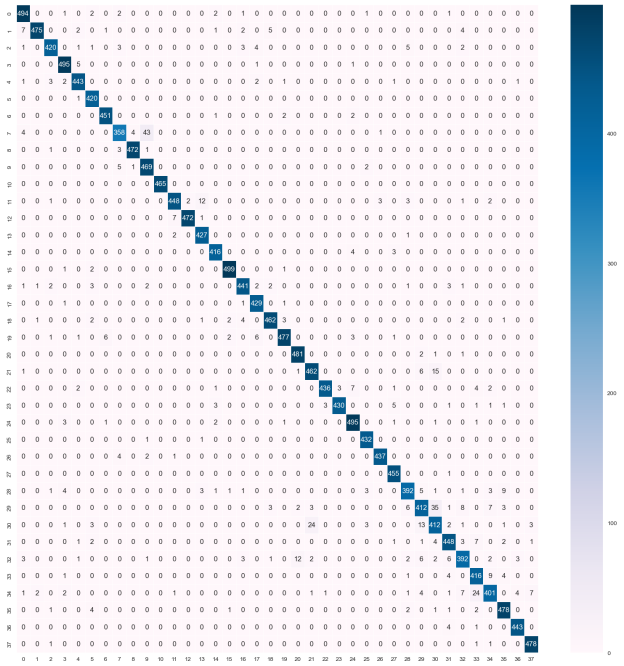


Fig. 25: Fully trained EfficientNet_B0 heatmap

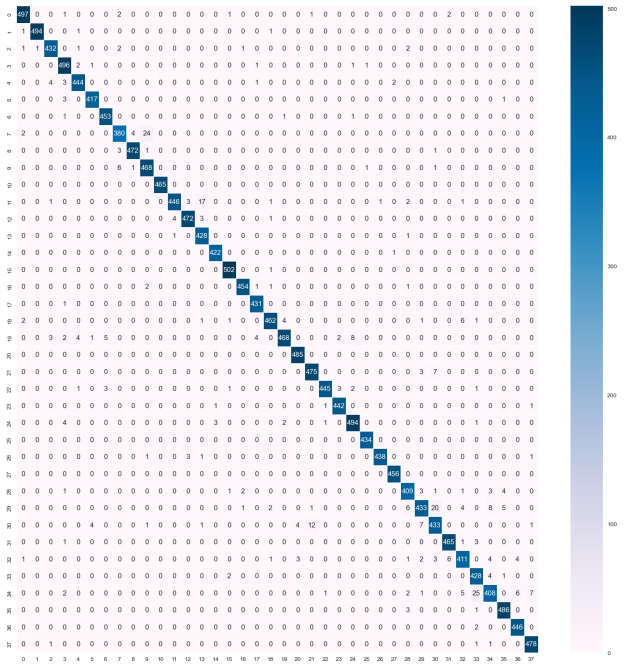


Fig. 26: Fully trained EfficientNet_B0 with tunning heatmap