

Research Article

Vol. 15, No. 3, 2025, p. 319-335

Detection and Classification of Some Diseases of Tomato Crops Using Transfer Learning

I. Ahmadi ^{1*}

1- Department of Genetics and Plant Production Engineering, Institute of Agriculture, Water, Food and Nutraceuticals, Isf. C., Islamic Azad University, Isfahan, Iran

(*- Corresponding Author Email: imanahmadi1358@iau.ac.ir)

Received: 14 June 2024

Revised: 13 July 2024

Accepted: 29 July 2024

Available Online: 31 May 2025

How to cite this article:

Ahmadi, I. (2025). Detection and Classification of Some Diseases of Tomato Crops Using Transfer Learning. *Journal of Agricultural Machinery*, 15(3), 319-335. <https://doi.org/10.22067/jam.2024.88500.1258>

Abstract

In the context of plant diseases, the selection of appropriate preventive measures, such as correct pesticide application, is only possible when plant diseases have been diagnosed quickly and accurately. In this study, a transfer learning model based on the pre-trained EfficientNet model was implemented to detect and classify some diseases in tomato crops, using an augmented training dataset of 2340 images of tomato plants. The study's findings indicate that during the model's validation phase, the rate of image categorization was roughly 5 fps (frames per second), which makes sense for a deep learning model operating on a laptop computer equipped with a standard CPU. Furthermore, the model was learned well because increasing the number of epochs no longer improved its accuracy. After all, the curves of the train and test accuracies, as well as the losses versus epoch numbers, remained largely horizontal for epoch numbers greater than 20. Notably, the highest coefficient of variation across these four cases was only 7%. Furthermore, the cells of the primary diagonal of the confusion matrix were filled with larger numbers in comparison with the values of the other cells; precisely, 88.8%, 7.7%, and 3.3% of the remaining cells of the matrix (cells of the primary diagonal excluded) were filled with 0, 1, and 2, respectively. The model's performance metrics are: sensitivity 85%, specificity 98%, precision 86%, F1-score 84%, and accuracy 85%.

Keywords: Confusion Matrix, EfficientNet Model, Model Accuracy

Introduction


In the context of plant diseases, the selection of appropriate preventive measures, such as correct pesticide application, is only possible when plant diseases have been diagnosed quickly and accurately. If plant diseases are not controlled on time, a significant reduction in crop quantity and quality will result.

One of the methods used to detect plant

diseases is through human observation of disease symptoms on different parts of the plant. Since different diseases may have similar symptoms, an amateur inspector may falsely diagnose a disease, and the wrong chemical may be chosen to cure the disease (Shah, Harshadkumar, Prajapati, & Dabhi, 2016); however, machine learning (ML) techniques such as feature extraction and classification can help an amateur inspector in this regard (Jena, Sethy, & Behera, 2021). Ali, Bachik, Muhadi, Tuan Yusof, and Gomes (2019) highlighted that utilizing machine learning techniques to evaluate plant diseases leads to enhanced precision, reliability, and accuracy. The inability of machine learning



©2025 The author(s). This is an open access article distributed under [Creative Commons Attribution 4.0 International License \(CC BY 4.0\)](https://creativecommons.org/licenses/by/4.0/).

 <https://doi.org/10.22067/jam.2024.88500.1258>

techniques to reliably detect subtle indications of plant illnesses and early-stage disease detection, and the requirement for complex and high-resolution images, are drawbacks of using feature extraction approaches for plant disease diagnosis (Anjna, Sood, & Singh, 2020; Genaev *et al.*, 2021).

Using deep learning (DL) techniques such as Convolutional Neural Networks (CNNs) and Deep Belief Networks (DBNs) can overcome the limitations of traditional ML methods (Khan, Khan, Albattah, & Qamar, 2021; Liu & Wang, 2021); however, DL models need high computational power, a limiting factor for some applications.

Another Artificial Intelligence (AI) model that has been recently used for plant disease detection is a model based on the transfer learning (TL) concept. In the TL paradigm, a pre-trained DL model is used to develop a custom model on another dataset that has the same function as the first pre-trained model. In other words, TL entails refining pre-trained models using a custom dataset to boost the performance of a model developed specifically for that dataset.

The accuracies obtained in studies conducted on plant disease identification using AI methods range from above 50% to nearly 100%. For example, Burhan, Minhas, Tariq, and Nabeel Hassan (2020) performed a study on five different modern DL models to identify rice diseases and obtained accuracies of 70.42%, 73.6%, 51.99%, 61.6%, and 87.79% on VGG16, VGG19, ResNet50, ResNet50v2, and ResNet101v2 models, respectively. Agarwal, Singh, Arjaria, Sinha, and Gupta (2020) carried out a study to classify 9 groups of tomato diseases and obtained accuracy values of 63.7%, 63.4%, and 90% using InceptionV3, MobileNet, and VGG16 models, respectively. Batool, Hyder, Rahim, Waheed, and Asghar (2020) aimed to detect and classify tomato leaf diseases using a training dataset of 450 images. Results showed that among the models used, the AlexNet model was superior, with an accuracy of 76.1%. Kirola *et al.* (2022) compared the performance of five ML models and one DL

model for plant disease prediction. They stated that the accuracies of ML models such as LR, SVM, KNN, RF, and NB were 71.89%, 75.76%, 82.17%, 97.12%, and 81.12%, respectively; while the accuracy of the deep learning CNN model was 98.43%. Bharali, Bhuyan, and Boruah (2019) developed a new architecture called plant disease detection neural network (PDDNN). An 86% accuracy was obtained for detecting diseases of maize, grape, apple, and tomato plants using this architecture. Balaji *et al.* (2023) utilized an enhanced CNN technique to detect rice diseases. For the TL, CNN+TL, ANN, and ECNN+GA models, they attained accuracies of 80%, 85%, 90%, and 95%, respectively. A fully connected CNN method was employed by Upadhyay and Kumar (2022) to identify three rice illnesses. Four thousand photos of each sick leaf and four thousand photos of healthy rice leaves were used to train the model. This study's accuracy rate was 99.7%.

To sum up, the development of a deep learning model for diagnosing some groups of tomato diseases, in order to assist in the curing process of diseased plants, is the main challenge followed in this study. Another challenge of this study is the lack of modern hardware resources required for running a deep learning model. In this study, it is necessary to run the model on a regular computer with a CPU instead of a modern GPU. It appears that the only deep learning model that may satisfy these criteria is the transfer learning (TL) model. As a result, a TL model based on the EfficientNet pre-trained model was implemented to distinguish between healthy tomatoes and nine groups of diseased plants. The original small custom dataset contained 320 images of tomato plants, divided into 234 images for the training dataset and 86 images for the test dataset; however, the training dataset was augmented, and the final number of images in it was increased to 2340 images. The following justifies the use of EfficientNet as a pre-trained model: EfficientNet emerges as a superior model for image classification tasks due to its efficient network structure and

ingenious mechanisms. It utilizes a triplet attention mechanism to enhance feature expression and acquire channel and spatial attention information, leading to improved accuracy. Additionally, the model enhances transfer learning by loading pre-trained parameters, accelerating convergence, and reducing training time, resulting in strong robustness and generalization ability. These specifications make EfficientNet a powerful choice for image classification tasks, surpassing other models in accuracy and efficiency (Huang, Su, Wu, & Chen, 2023).

Materials and Methods

Preparation of raw images

Nine predominant diseases of the tomato plant (*Solanum lycopersicum*) along with healthy crops, were considered as targets for diagnosis. The names of these diseases served as the classifier groups' names: Anthracnose, Bacterial_Speck, Blossom_End_Rot, Buckeye_Rot, Damping_Off, Gray_Wall, Healthy_Tomato, Leaf_Mold, Southern_Blight, and Tomato_Pith_Necrosis. Some of the pictures belonging to each group were considered for developing the model. While the total number of images was 2426, they were divided into 2340 pictures in the train folder and 86 pictures in the test folder. The exact number of pictures in the train and test datasets of each classifier group are as follows: Anthracnose (300, 11), Bacterial_Speck (170, 6), Blossom_End_Rot (350, 13), Buckeye_Rot (200, 7), Damping_Off (220, 9), Gray_Wall (120, 5), Healthy_Tomato (230, 10), Leaf_Mold (320, 11), Southern_Blight (140, 4), and Tomato_Pith_Necrosis (290, 10). The first number in the parentheses indicates the number of images in the training dataset for that classifier group, while the second number indicates the number of images in the test dataset. Images of the "train" folder will be used for training the model (i.e. changing weights of the model in the convergence direction so that the model predictions on train images match reality), while the unseen images of the "test" folder will be used to test

the goodness of those weights for diagnosing tomato diseases.

Model development procedure

Development of the model was conducted using code written in the PyCharm environment, using the PyTorch package of the Python programming language. The codes are attached to this paper as an appendix. Some functions in the PyTorch package make it suitable for creating models based on the transfer learning concept.

The architecture of the TL model to classify 10 groups of tomato diseases is shown in Table 1.

As shown, the parameters of the feature extraction section are not trainable during the model's training process, and only the parameters of the classifier section are trainable. This leads to a significant decrease in the model's training time. Numerically, the number of trainable parameters decreased from 4,020,358 parameters of the EfficientNet model to 12,810 parameters of the model based on the transfer learning concept. This feature of the TL model allows it to run efficiently on a regular computer with a standard CPU in a reasonable time.

Other informative data about the TL model are summarized in Table 2.

Model evaluation criteria

Sensitivity (Recall): This criterion measures how well a model can detect positive instances; in mathematical terms, it can be calculated using the formula:

$$\text{Sensitivity (Recall)} = \frac{TP}{TP+FN} \quad (1)$$

Where TP is the number of instances that are truly predicted as positive, and FN is the number of instances that are falsely predicted as negative.

Specificity: This criterion measures the model's ability to predict true negatives in each of the available categories. In mathematical terms, it can be calculated using the formula:

$$\text{Specificity} = \frac{TN}{TN+FP} \quad (2)$$

Table 1- The architecture of the TL model based on the EfficientNet pre-trained model

Layer (type (var_name))	Input Shape	Output Shape	Param #	Trainable
EfficientNet (EfficientNet)	[32, 3, 224, 224]	[32, 10]	--	Partial
└ Sequential (features)	[32, 3, 224, 224]	[32, 1280, 7, 7]	--	False
└ Conv2dNormActivation (0)	[32, 3, 224, 224]	[32, 32, 112, 112]	--	False
└ Conv2d (0)	[32, 3, 224, 224]	[32, 32, 112, 112]	(864)	False
└ BatchNorm2d (1)	[32, 32, 112, 112]	[32, 32, 112, 112]	(64)	False
└ SiLU (2)	[32, 32, 112, 112]	[32, 32, 112, 112]	--	--
└ Sequential (1)	[32, 32, 112, 112]	[32, 16, 112, 112]	--	False
└ MBConv (0)	[32, 32, 112, 112]	[32, 16, 112, 112]	(1,448)	False
└ Sequential (2)	[32, 16, 112, 112]	[32, 24, 56, 56]	--	False
└ MBConv (0)	[32, 16, 112, 112]	[32, 24, 56, 56]	(6,004)	False
└ MBConv (1)	[32, 24, 56, 56]	[32, 24, 56, 56]	(10,710)	False
└ Sequential (3)	[32, 24, 56, 56]	[32, 40, 28, 28]	--	False
└ MBConv (0)	[32, 24, 56, 56]	[32, 40, 28, 28]	(15,350)	False
└ MBConv (1)	[32, 40, 28, 28]	[32, 40, 28, 28]	(31,290)	False
└ Sequential (4)	[32, 40, 28, 28]	[32, 80, 14, 14]	--	False
└ MBConv (0)	[32, 40, 28, 28]	[32, 80, 14, 14]	(37,130)	False
└ MBConv (1)	[32, 80, 14, 14]	[32, 80, 14, 14]	(102,900)	False
└ MBConv (2)	[32, 80, 14, 14]	[32, 80, 14, 14]	(102,900)	False
└ Sequential (5)	[32, 80, 14, 14]	[32, 112, 14, 14]	--	False
└ MBConv (0)	[32, 80, 14, 14]	[32, 112, 14, 14]	(126,004)	False
└ MBConv (1)	[32, 112, 14, 14]	[32, 112, 14, 14]	(208,572)	False
└ MBConv (2)	[32, 112, 14, 14]	[32, 112, 14, 14]	(208,572)	False
└ Sequential (6)	[32, 112, 14, 14]	[32, 192, 7, 7]	--	False
└ MBConv (0)	[32, 112, 14, 14]	[32, 192, 7, 7]	(262,492)	False
└ MBConv (1)	[32, 192, 7, 7]	[32, 192, 7, 7]	(587,952)	False
└ MBConv (2)	[32, 192, 7, 7]	[32, 192, 7, 7]	(587,952)	False
└ MBConv (3)	[32, 192, 7, 7]	[32, 192, 7, 7]	(587,952)	False
└ Sequential (7)	[32, 192, 7, 7]	[32, 320, 7, 7]	--	False
└ MBConv (0)	[32, 192, 7, 7]	[32, 320, 7, 7]	(717,232)	False
└ Conv2dNormActivation (8)	[32, 320, 7, 7]	[32, 1280, 7, 7]	--	False
└ Conv2d (0)	[32, 320, 7, 7]	[32, 1280, 7, 7]	(409,600)	False
└ BatchNorm2d (1)	[32, 1280, 7, 7]	[32, 1280, 7, 7]	(2,560)	False
└ SiLU (2)	[32, 1280, 7, 7]	[32, 1280, 7, 7]	--	--
└ AdaptiveAvgPool2d (avgpool)	[32, 1280, 7, 7]	[32, 1280, 1, 1]	--	--
└ Sequential (classifier)	[32, 1280]	[32, 10]	--	True
└ Dropout (0)	[32, 1280]	[32, 1280]	--	--
└ Linear (1)	[32, 1280]	[32, 10]	12,810	True

Total params: 4,020,358
 Trainable params: 12,810
 Non-trainable params: 4,007,548
 Total mult-adds (Units.GIGABYTES): 12.31

Input size (MB): 19.27
 Forward/backward pass size (MB): 3452.09
 Params size (MB): 16.08
 Estimated Total Size (MB): 3487.44

Table 2- Complementary information about the TL model

Parameter	Setting
Training epoch	30
Batch size	32
optimizer	Adam
Learning rate	0.001
Loss function	Cross-Entropy

where TN is the number of instances that are truly predicted as negative, and FP is the number of instances that are falsely predicted

as positive.

Precision: This criterion measures the ratio of the number of instances that are predicted as true positives to the total number of instances predicted as positive. In mathematical terms, it can be calculated using the formula:

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

F1 score: The F1 score is calculated by taking the harmonic average of recall and precision. In mathematical terms, we have:

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (4)$$

If either precision or recall is zero, then the F1 score will be zero, but the arithmetic average of precision and recall will be a number greater than zero.

Accuracy: This criterion measures the ratio of the number of true predictions made by the model to the total number of predictions. In mathematical terms, we have:

$$Accuracy = \frac{\sum_{i=1}^n (TP_i)}{\sum_{i=1}^n (TP_i + FP_i)} \quad (5)$$

Where n is the number of classes; in this study, n is 10. During model training, this criterion was calculated after each epoch (there are 30 epochs during model training in this study), and then accuracy and loss values were charted against the corresponding epoch number. These charts can be used to be sure about the number of epochs for model training; in other words, if the curve of accuracy or loss versus epoch number remains horizontal for a few epochs, it is an indication for the completion of the model's training phase.

Confusion matrix: To obtain a complete understanding of where a false prediction has taken place, and with which group the prediction has been confused, a confusion

matrix will be used. In this paper, a 10×10 confusion matrix was developed to show the location of false tomato disease predictions. In a confusion matrix, the greater the concentration of large numbers in the cells of the main diagonal of the matrix (the more zeros or small numbers in the other cells of the matrix), the better. The values of TP, FP, FN, and TN can be obtained from the confusion matrix. The values of the cells located on the main diagonal of the matrix comprise the TP values. If the TP value of each class is subtracted from the sum of cells for the corresponding column, the FP value for that class will result. On the other hand, if the TP value of each class is subtracted from the sum of the cells for the corresponding row, the FN value of that class will be obtained. Finally, the TN values can be obtained by subtracting the sum of the TP, FP, and FN values from the number of pictures in the test dataset.

Results and Discussion

Accuracy values versus corresponding epoch numbers: Figure 1 shows the train and test accuracies, as well as the losses of the model versus the corresponding epoch numbers.

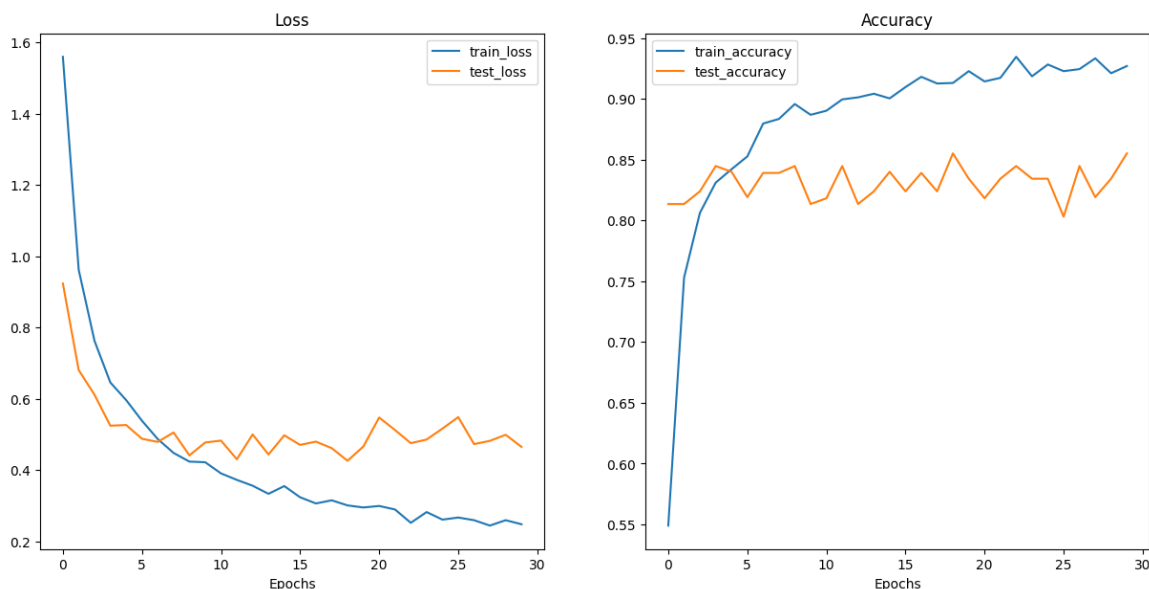


Fig. 1. Train and test losses and accuracies versus corresponding epoch number

The first point inferred from Figure 1 is that

because the curves of train and test accuracies

have been approximately horizontal for epoch numbers greater than 20, the model has learned well, and increasing the epoch number cannot further increase model accuracy. Numerically, by recording values of model accuracies and losses in the training and testing phases for epoch numbers greater than 20, the worst coefficient of variation for these four cases was 7%. Another point that can be

obtained from this chart is that since the distance between train and test accuracies is about 10%, and this situation has taken place for train accuracies near 95%, therefore, the model shows neither signs of under fitting nor over fitting.

Confusion matrix of the test instances:

Figure 2 shows the confusion matrix obtained for the 86 images in the test dataset.

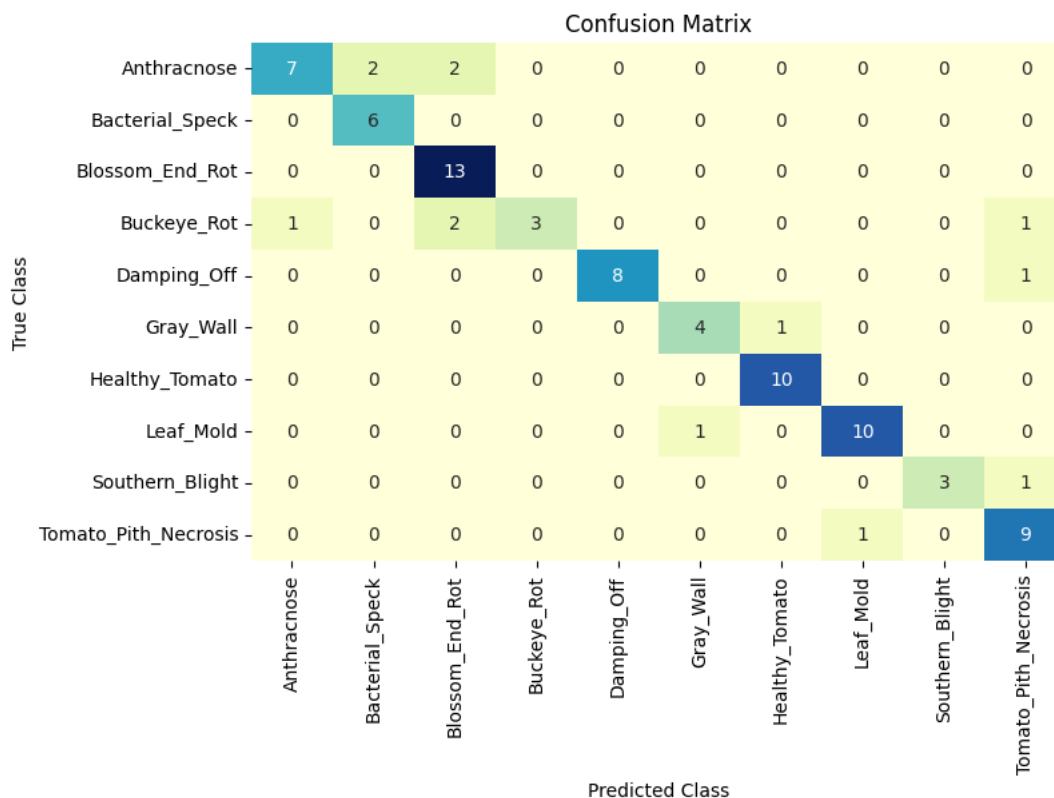


Fig. 2. Confusion matrix obtained in this study

This picture presents a matrix of real (ground truth) labels versus predicted ones. As shown, the elements along the primary diagonal of the matrix are filled with larger values compared to those in the other cells. The values of the primary diagonal of the matrix comprise the number of images in each disease's category that were predicted truly as positive; in other words, these values are the *TP* values of classes. The *FP*, *TN*, and *FN* values can be obtained from this matrix, too. Table 3 shows *TP*, *FP*, *TN*, and *FN* values of each category of tomato diseases considered in this study:

Calculation of sensitivity (recall), specificity, precision, F1 score, and accuracy of the model: The values presented in Table 3 can be used to obtain sensitivity (recall), specificity, precision, F1 score, and accuracy of the model. Table 4 shows the calculated values of the above-mentioned criteria.

The criteria values shown in Table 4 were calculated for each category of tomato diseases. To obtain corresponding criteria for the model, regular or weighted averaging can be carried out. Table 5 shows the sensitivity (recall), specificity, precision, and F1 score of

the model.

Table 3- Suitable data obtained from the confusion matrix

Disease category	TP	FP	FN	TN
Anthrachnose	7	1	4	74
Bacterial Speck	6	2	0	78
Blossom_End_Rot	13	4	0	69
Buckeye Rot	3	0	4	79
Damping_Off	8	0	1	77
Gray Wall	4	1	1	80
Healthy Tomato	10	1	0	75
Leaf_Mold	10	1	1	74
Southern Blight	3	0	1	82
Tomato_Pith_Necrosis	9	3	1	73

Table 4- Calculated values of sensitivity (recall), specificity, precision, and F1 score

Disease category	Sensitivity (recall)	Specificity	Precision	F1 score
Anthrachnose	0.64	0.99	0.88	0.74
Bacterial Speck	1	0.98	0.75	0.86
Blossom_End_Rot	1	0.95	0.76	0.86
Buckeye Rot	0.43	1	1	0.6
Damping_Off	0.89	1	1	0.94
Gray Wall	0.8	0.99	0.8	0.8
Healthy Tomato	1	0.99	0.91	0.95
Leaf_Mold	0.91	0.99	0.91	0.91
Southern Blight	0.75	1	1	0.86
Tomato_Pith_Necrosis	0.9	0.96	0.75	0.82

Table 5- Sensitivity (recall), specificity, precision, and F1 score of the model calculated from regular and weighted averaging

Average	Sensitivity (recall)	Specificity	Precision	F1 score
Regular	0.832	0.985	0.876	0.834
Weighted	0.85	0.982	0.868	0.841

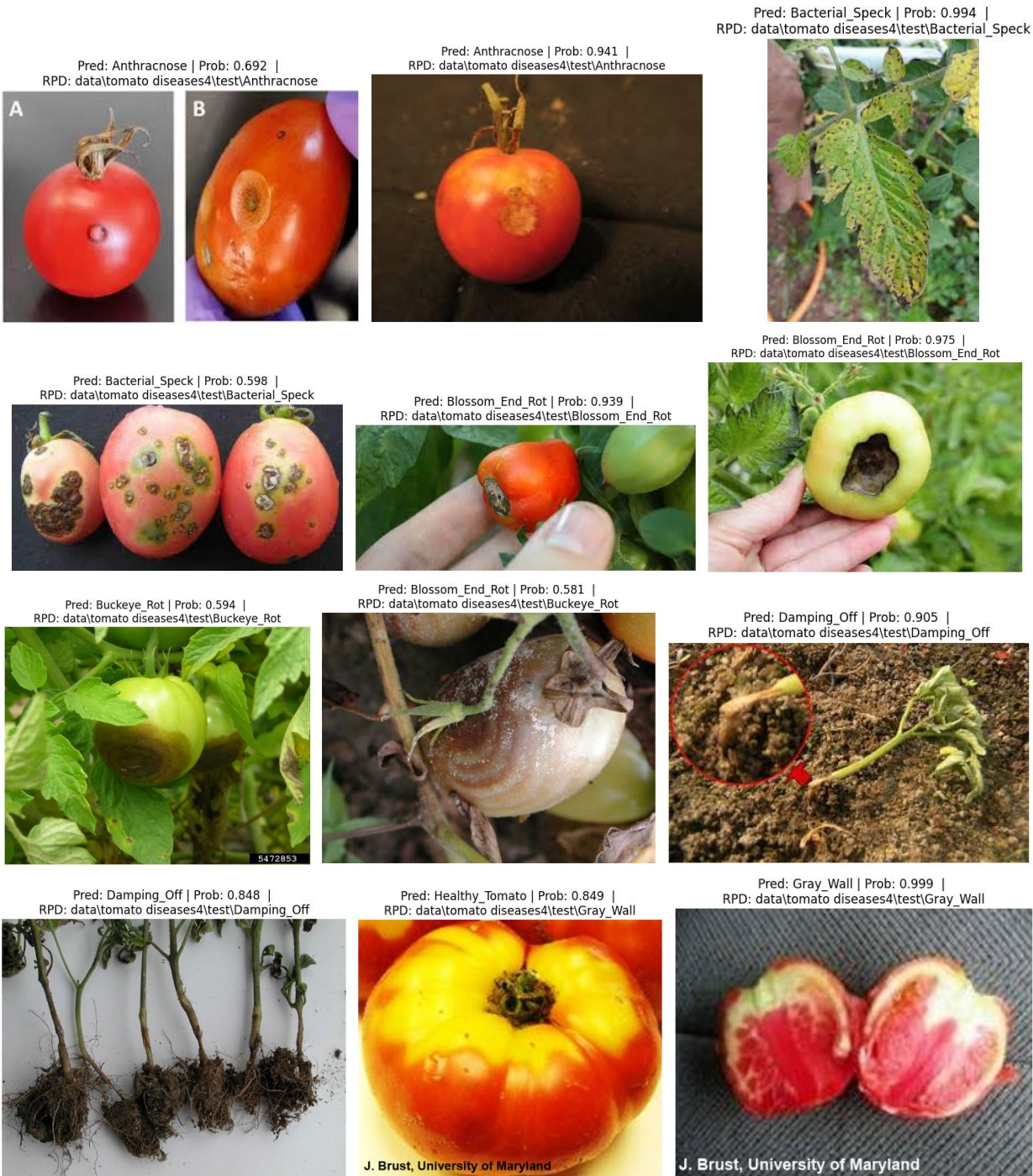
The accuracy of the model was obtained from the formula of *accuracy* =

$$\frac{\sum_{i=1}^{10}(TP_i)}{\sum_{i=1}^{10}(TP_i+FP_i)} = \frac{73}{86} = 0.85.$$

To assess the performance of the model visually, model predictions on some tomato images were considered. Figure 3 shows the predictions of the model on twenty randomly selected images in the test dataset.

Three acronyms are displayed on top of each image: Pred, Prob, and RPD. Pred is the acronym for prediction and shows the predicted group of the image. RPD is the acronym for real parent directory and shows the correct directory that the image belongs to. Prob is the acronym for probability and presents the probability value that the image belongs to the predicted group. In other words,

the classifying algorithm calculates ten probabilities for each image that represent the belonging probabilities of the image to each of the tomato disease groups, and declares the tomato disease group with the highest probability value as the image's predicted group. As shown in Fig. 3, 3 out of 20 predictions were incorrect, which indicates the error rate of 15% (i.e. the accuracy of 0.85). On the other hand, there is a trade-off between the performance of a machine learning method and the time spent or computational resources devoted to training the model. Although the time spent training the model in this study was about 4.5 hours, which is high, the model was trained on a computer without a GPU. Additionally, it took just 16 seconds to classify 86 images from the test dataset during the model's validation phase.



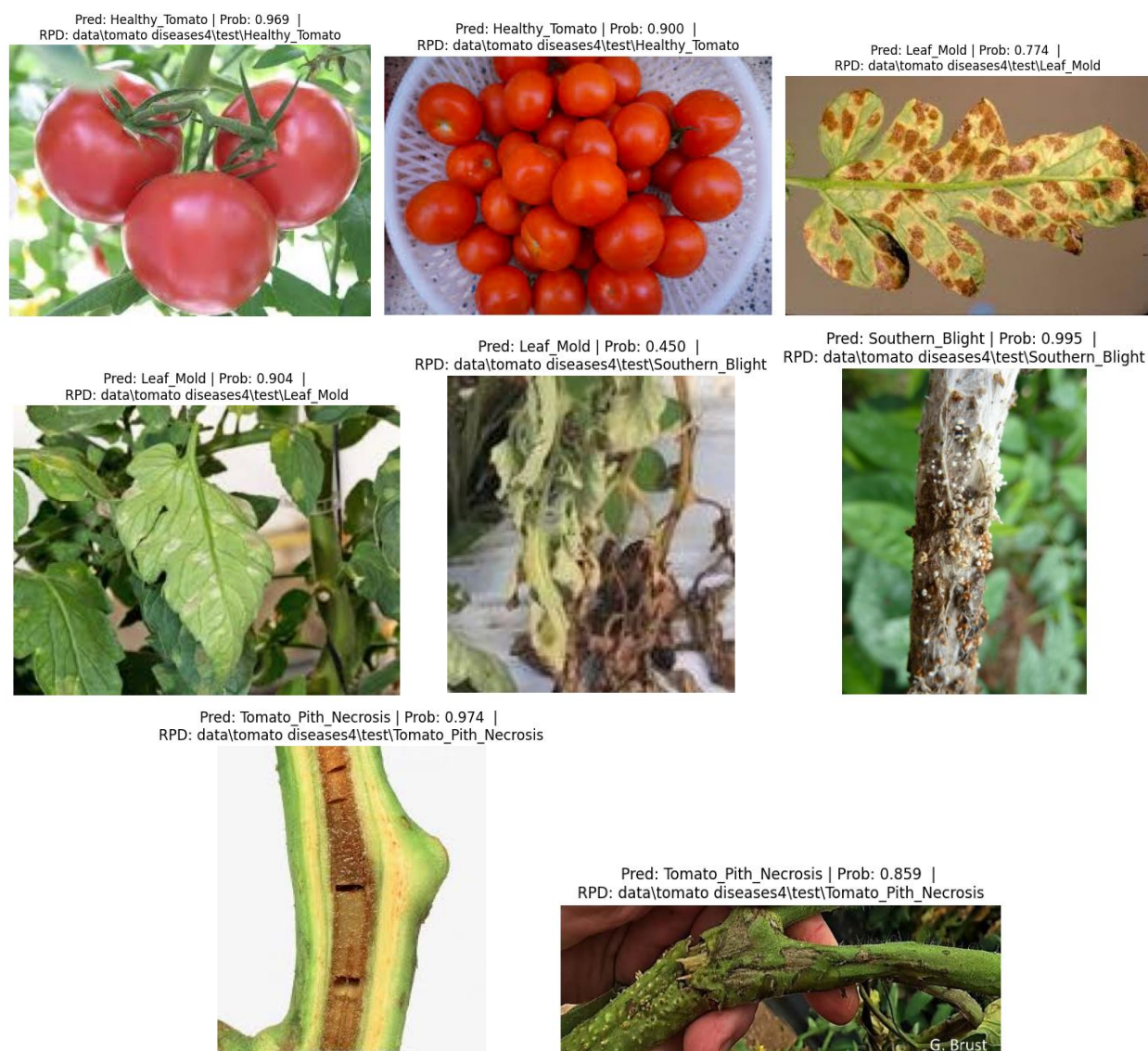


Fig. 3. Predictions of the model on some random images of tomato plants selected from the test dataset

This means that the model performed well during validation, achieving an image classification rate of about 5 frames per second (fps). Furthermore, the model's accuracy was comparable to some of the findings made by other researchers (Burhan *et al.*, 2020; Batool *et al.*, 2020); however, in some similar studies, higher accuracies were obtained (Jasim & Al-Tuwaijari, 2020; Guerrero-Ibanez & Reyes-Munoz, 2023; Ahmed & Yadav, 2023; Zhao, Peng, Liu, & Wu, 2021). To explain the observed differences, it should be noted that in the studies conducted by Jasim and Al-Tuwaijari (2020) and Guerrero-Ibanez and Reyes-Munoz (2023), they reached accuracies

higher than 98% by training complete CNN models, instead of transferring the weights of a pre-trained model to a TL model and only training the classifier section of the model. On the other hand, in the studies conducted by Zhao *et al.* (2021), and Ahmed and Yadav (2023), more images capturing only the leaves of plants were used to train the model. This differs from the proportion of images examined in this study. Therefore, it is necessary to consider all the conditions surrounding a model and not focus on a single criterion in order to have a fair judgment about the model's performance.

Conclusion

In conclusion, in this study, a computationally inexpensive TL model trained on a dataset containing a mixture of images such as leaf, fruit, and stem was used to discriminate between healthy tomatoes and nine groups of diseased plants. According to the obtained results, the model's overall

performance metrics are: sensitivity 85%, specificity 98%, precision 86%, F1-score 84%, and accuracy 85%, which is acceptable.

Conflict of Interest: The author declares no competing interests.

References

1. Agarwal, M., Singh, A., Arjaria, S., Sinha, A., & Gupta S. T. (2020). Tomato leaf disease detection using convolution neural network. *Procedia Computer Science*, 167, 293-301. <https://doi.org/10.1016/j.procs.2020.03.225>
2. Ahmed, I., & Yadav, P. K. (2023). A systematic analysis of machine learning and deep learning based approaches for identifying and diagnosing plant diseases. *Sustainable Operations and Computers*, 4, 96-104. <https://doi.org/10.1016/j.susoc.2023.03.001>
3. Ali, M. M., Bachik, N. A., Muhadi, N. A., Tuan Yusof, T. N., & Gomes, C. (2019). Nondestructive techniques of detecting plant diseases: A review. *Physiological and Molecular Plant Pathology*, 108, 101426. <https://doi.org/10.1016/j.pmpp.2019.101426>
4. Anjna, Sood, M., & Singh, P. K. (2020). Hybrid system for detection and classification of plant disease using qualitative texture features analysis. *Procedia Computer Science*, 167, 1056-1065. <https://doi.org/10.1016/j.procs.2020.03.404>
5. Balaji, V., Anushkannan, N. K., Narahari, Sujatha Canavoy, Rattan, Punam, Verma, Devvret, Awasthi, Deepak Kumar, Pandian, A. Anbarasa, Veeramanickam, M. R. M., Mulat, & Molla Bayih (2023). Deep transfer learning technique for multimodal disease classification in plant images. *Contrast Media & Molecular Imaging*, 5644727. <https://doi.org/10.1155/2023/5644727>
6. Batool, A., Hyder, S. B., Rahim, A., Waheed, N., & Asghar, M.A. (2020). *Classification and identification of tomato leaf disease using deep neural network*. International Conference on Engineering and Emerging Technologies (ICEET). <https://doi.org/10.1109/ICEET48479.2020.9048207>
7. Burhan, S. A., Minhas, D. S., Tariq, D. A., & Nabeel, H. M. (2020). *Comparative study of deep learning algorithms for disease and pest detection in rice crops*. 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). <https://doi.org/10.1109/ECAI50035.2020.9223239>
8. Genaev, M. A., Skolotneva, E. S., Gulyaeva, E. I., Orlova, E. A., Bechtold, N. P., & Afonnikov, D. A. (2021). Image-based wheat fungi diseases identification by deep learning. *Plants*, 10(8), 1-21. <https://doi.org/10.3390/plants10081500>
9. Guerrero-Ibanez, A., & Reyes-Munoz, A. (2023). Monitoring tomato leaf disease through convolutional neural networks. *Electron*, 12(1), 1-15. <https://doi.org/10.3390/electronics12010229>
10. Bharali, P., Bhuyan, C., & Boruah, A. (2019). Plant Disease Detection by Leaf Image Classification Using Convolutional Neural Network. In: Gani, A., Das, P., Kharb, L., Chahal, D. (eds) Information, Communication and Computing Technology. ICICCT 2019. *Communications in Computer and Information Science*, vol 1025. Springer, Singapore. https://doi.org/10.1007/978-981-15-1384-8_16
11. Huang, Z., Su, L., Wu, J., & Chen, Y. (2023). Rock Image Classification Based on EfficientNet and Triplet Attention Mechanism. *Applied Science Letters*, 13, 3180.

<https://doi.org/10.3390/app13053180>

12. Jasim, M. A., & Al-Tuwaijari, J. M. (2020). Plant leaf diseases detection and classification using image processing and deep learning techniques. *International Conference on Computer Science and Software Engineering*. <https://doi.org/10.1109/CSASE48920.2020.9142097>
13. Jena, L., Sethy, P. K., & Behera, S. K. (2021). *Identification of wheat grain using geometrical feature and machine learning*. In: 2nd international conference for emerging technology (INCET) (pp. 1_6).
14. Khan, R. U., Khan, K., Albattah, W., & Qamar, A.M. (2021). Image-based detection of plant diseases: from classical machine learning to deep learning journey. *Wireless Communications and Mobile Computing*, 1-13. <https://doi.org/10.1155/2021/5541859>
15. Kirola, M., Joshi, K., Chaudhary, S., Singh, N., Anandaram, H., & Gupta, A. (2022). *Plants diseases prediction framework: a imagebased system using deep learning*. Proc IEEE World Conf Appl Intell Comput. <https://doi.org/10.1109/AIC55036.2022.9848899>
16. Liu, J., & Wang, X. (2021). Plant diseases and pests detection based on deep learning: a review. *Plant Methods*, 17(1), 1-18. <https://doi.org/10.1186/s13007-021-00722-9>
17. Shah, J. P., Harshadkumar, B., Prajapati, V. K., & Dabhi. (2016). *A survey on detection and classification of rice plant diseases*. In: IEEE international conference on current trends in advanced computing (ICCTAC).
18. Upadhyay, S. K., & Kumar, A. (2022). A novel approach for rice plant diseases classification with deep convolutional neural network. *International Journal of Information Technology*, 14(1):185-99. <https://doi.org/10.1007/s41870-021-00817-5>
19. Zhao, S., Peng, Y., Liu, J., & Wu, S. (2021). Tomato leaf disease diagnosis based on improved convolution neural network by attention module. *Agriculture*, <https://doi.org/10.3390/agriculture11070651>

Appendix: Python codes used in this study¹:

This script aims to create augmented images from one image to create a larger dataset for our transfer learning model

```
import PIL
import torch
from PIL import Image
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
import sys
import torchvision.transforms as T
import os

#torch.transforms

#grayscale
grayscale_transform = T.Grayscale(3)

#random rotation
random_rotation_transformation_45 = T.RandomRotation(45)
random_rotation_transformation_85 = T.RandomRotation(85)
```

1- The codes were developed based on the materials presented in the following websites:

<https://anushsom.medium.com/image-augmentation-for-creating-datasets-using-pytorch-for-dummies-by-a-dummy-a7c2b08c5bcb>

https://www.learnpytorch.io/06_pytorch_transfer_learning/

```
#Gaussian Blur
gaussian_blur_transformation_13 = T.GaussianBlur(kernel_size = (7,13), sigma = (6 , 9))
gaussian_blur_transformation_56 = T.GaussianBlur(kernel_size = (7,13), sigma = (5 , 8))

#Gaussian Noise
def addnoise(input_image, noise_factor = 0.3):
    inputs = T.ToTensor()(input_image)
    noisy = inputs + torch.rand_like(inputs) * noise_factor
    noisy = torch.clip (noisy,0,1.)
    output_image = T.ToPILImage()
    image = output_image(noisy)
    return image

#Colour Jitter
colour_jitter_transformation_1 = T.ColorJitter(brightness=(0.5,1.5),contrast=(3),saturation=(0.3,1.5),hue=(-0.1,0.1))
colour_jitter_transformation_3 = T.ColorJitter(brightness=(0.5,1.5),contrast=(2),saturation=(1.4),hue=(-0.1,0.5))

#Random invert
random_invert_transform = T.RandomInvert()

#Main function that calls all the above functions to create 9 augmented images from one image

def augment_image(img_path):

    #orig_image
    orig_img = Image.open(Path(img_path))

    #grayscale
    grayscaled_image=grayscale_transform(orig_img)
    #grayscaled_image.show()

    #random rotation
    random_rotation_transformation_45_image=random_rotation_transformation_45(orig_img)
    #random_rotation_transformation_45_image.show()

    random_rotation_transformation_85_image=random_rotation_transformation_85(orig_img)
    #random_rotation_transformation_85_image.show()

    #Gaussian Blur
    gaussian_blurred_image_13_image = gaussian_blur_transformation_13(orig_img)
    #gaussian_blurred_image_13_image.show()

    gaussian_blurred_image_56_image = gaussian_blur_transformation_56(orig_img)
    #gaussian_blurred_image_56_image.show()

    #Gaussian Noise
    gaussian_image_3 = addnoise(orig_img)
    #gaussian_image_3.show()

    gaussian_image_9 = addnoise(orig_img,0.9)
    #gaussian_image_9.show()

    #Color Jitter
    colour_jitter_image_1 = colour_jitter_transformation_1(orig_img)
    #colour_jitter_image_1.show()

    colour_jitter_image_3 = colour_jitter_transformation_3(orig_img)
```

```
#colour_jitter_image_3.show()

return [orig_img,grayscaled_image,random_rotation_transformation_45_image,
random_rotation_transformation_85_image,gaussian_blurred_image_13_image,gaussian_blurred_image_56_image,gaussian_image_3, gaussian_image_9,colour_jitter_image_1, colour_jitter_image_3]

#augmented_images = augment_image(orig_img_path)

def creating_file_with_augmented_images(file_path_master_dataset,file_path_augmented_images):

    master_dataset_folder = file_path_master_dataset
    files_in_master_dataset = os.listdir(file_path_master_dataset)
    augmented_images_folder = file_path_augmented_images

    counter=0

    for element in files_in_master_dataset:
        os.mkdir(f"{augmented_images_folder}/{element}")
        images_in_folder= os.listdir(f"{master_dataset_folder}/{element}")
        counter = counter+1
        counter2 = 0
        for image in images_in_folder:
            counter
            required_images = augment_image(f"{master_dataset_folder}/{element}/{image}")
            counter2=counter2+1
            counter3 = 0
            for augmented_image in required_images:
                counter3 = counter3 +1
                augmented_image
            augmented_image.save(f"{augmented_images_folder}/{element}/{counter}_{counter2}_{counter3}_{image}")

    """images = augment_image("dog.png")

    for element in images:
        element.show()"""

#augmented dataset path
augmented_dataset = "C:\\Users\\acer\\PycharmProjects\\pythonProject2\\augmented_tomato"

# master dataset path
master_dataset = "C:\\Users\\acer\\PycharmProjects\\pythonProject2\\tomato"

# run the program

creating_file_with_augmented_images(master_dataset,augmented_dataset)

# This script aims to perform the main goal of the study i.e. tomato crop diseases classification

import torch
import torchvision
import matplotlib.pyplot as plt
from torch import nn
from torchvision import transforms
!pip install going_modular
import going_modular
try:
    from torchinfo import summary
except:
    print("[INFO] Couldn't find torchinfo... installing it.")
```



```
!pip3 install -q torchinfo
from torchinfo import summary
try:
    from going_modular.going_modular import data_setup, engine
except:
    print("[INFO] Couldn't find going_modular scripts... downloading them from GitHub.")
    !git clone https://github.com/mrdbourke/pytorch-deep-learning
    !mv pytorch-deep-learning/going_modular .
    !rm -rf pytorch-deep-learning
    from going_modular.going_modular import data_setup, engine
device = "cuda" if torch.cuda.is_available() else "cpu"
device
import os
import zipfile
from pathlib import Path
import requests
data_path = Path("data/")
image_path = data_path / "tomato_diseases"
train_dir = image_path / "train"
test_dir = image_path / "test"
weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT
auto_transforms = weights.transforms()
auto_transforms
train_dataloader, test_dataloader, class_names = data_setup.create_dataloaders(train_dir=train_dir,
                                                                              test_dir=test_dir,
                                                                              transform=auto_transforms,
                                                                              batch_size=32)

train_dataloader, test_dataloader, class_names
weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT
model = torchvision.models.efficientnet_b0(weights=weights).to(device)
summary(model=model,
        input_size=(32, 3, 224, 224),
        col_names=["input_size", "output_size", "num_params", "trainable"],
        col_width=10,
        row_settings=["var_names"]
)
for param in model.features.parameters():
    param.requires_grad = False
torch.manual_seed(42)
torch.cuda.manual_seed(42)
output_shape = len(class_names)
model.classifier = torch.nn.Sequential(
    torch.nn.Dropout(p=0.2, inplace=True),
    torch.nn.Linear(in_features=1280,
                    out_features=output_shape,
                    bias=True)).to(device)
summary(model,
        input_size=(32, 3, 224, 224),
        verbose=0,
        col_names=["input_size", "output_size", "num_params", "trainable"],
        col_width=20,
        row_settings=["var_names"]
)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
torch.manual_seed(42)
torch.cuda.manual_seed(42)
from timeit import default_timer as timer
start_time = timer()
```

```

results = engine.train(model=model,
                        train_dataloader=train_dataloader,
                        test_dataloader=test_dataloader,
                        optimizer=optimizer,
                        loss_fn=loss_fn,
                        epochs=5,
                        device=device)
end_time = timer()
print(f"[INFO] Total training time: {end_time-start_time:.3f} seconds")
try:
    from helper_functions import plot_loss_curves
except:
    print("[INFO] Couldn't find helper_functions.py, downloading...")
    with open("helper_functions.py", "wb") as f:
        import requests
        request = requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-deep-learning/main/helper_functions.py")
        f.write(request.content)
    from helper_functions import plot_loss_curves
plot_loss_curves(results)
from typing import List, Tuple
from pathlib import Path
def get_folder(dataset):
    parent = Path(dataset).parent
    if "." in parent.name:
        return str(parent.parent)
    return str(parent)
from PIL import Image
def pred_and_plot_image(model: torch.nn.Module,
                        image_path: str,
                        class_names: List[str],
                        image_size: Tuple[int, int] = (224, 224),
                        transform: torchvision.transforms = None,
                        device: torch.device=device):
    img = Image.open(image_path)
    if transform is not None:
        image_transform = transform
    else:
        image_transform = transforms.Compose([
            transforms.Resize(image_size),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                std=[0.229, 0.224, 0.225]),
        ])
    model.to(device)
    model.eval()
    with torch.inference_mode():
        transformed_image = image_transform(img).unsqueeze(dim=0)
        target_image_pred = model(transformed_image.to(device))
        target_image_pred_probs = torch.softmax(target_image_pred, dim=1)
        target_image_pred_label = torch.argmax(target_image_pred_probs, dim=1)
        plt.figure()
        plt.imshow(img)
        plt.title(f"Pred: {class_names[target_image_pred_label]} | Prob: {target_image_pred_probs.max():.3f} | Real
Parent Directory: {get_folder(image_path)}")
        plt.axis(False);
    import random
    num_images_to_plot = 30
    test_image_path_list = list(Path(test_dir).glob("*/*.jpg"))

```

```
test_image_path_sample = random.sample(population=test_image_path_list,
                                         k=num_images_to_plot)
for image_path in test_image_path_sample:
    pred_and_plot_image(model=model,
                        image_path=image_path,
                        class_names=class_names,
                        image_size=(224, 224))
```

تشخیص و دسته‌بندی تعدادی از بیماری‌های گوجه‌فرنگی با استفاده از یادگیری انتقالی

ایمان احمدی^{۱*}

تاریخ دریافت: ۱۴۰۳/۰۳/۲۵

تاریخ پذیرش: ۱۴۰۳/۰۵/۰۸

چکیده

در موضوع بیماری‌های گیاهی، انتخاب روش‌های پیشگیری مناسب مانند کاربرد علف‌کش‌ها تنها زمانی امکان‌پذیر است که نوع بیماری به سرعت و به دقت تشخیص داده شود. در این مطالعه یک مدل یادگیری انتقالی بر مبنای مدل از پیش آموزش دیده EfficientNet به منظور تشخیص و دسته‌بندی برخی از بیماری‌های گوجه‌فرنگی با استفاده از ۲۳۴۰ تصویر از گیاه گوجه‌فرنگی به عنوان پایگاه تصاویر آموزشی توسعه یافت. بر مبنای نتایج این مطالعه در مرحله اعتبارسنجی مدل، نرخ دسته‌بندی تصاویر در حدود ۵ fps (۵ فریم در ثانیه) بود که برای مدل یادگیری عمیقی که روی رایانه مجهز به CPU معمولی اجرا می‌شود، منطقی است. به علاوه چون منحنی‌های صحت و هزینه مربوط به مراحل آموزش و آزمایش در عده‌های دوره بیش از ۲۰ تقریباً افقی شده بودند (از نظر عددی، بدترین ضریب تغییرات داده‌های مربوط به این چهار حالت ۷٪ بود)، مدل به خوبی آموزش دیده است و افزایش عدد دوره به افزایش صحت مدل منجر نخواهد شد. به علاوه سلول‌های واقع در قطر اصلی ماتریس در هم‌ریختگی مربوط به عملکرد مدل روی تصاویر آزمون با اعداد بزرگتری در مقایسه با محتوای سایر سلول‌های ماتریس پر شده بودند، به طور دقیق ۸/۸۸٪، ۷/۷٪، و ۳/۳٪ از سلول‌های باقیمانده (جایی که سلول‌های مربوط به قطر اصلی ماتریس کنار گذاشته شده بودند) به ترتیب با اعداد ۰ و ۱ و ۲ پر شده بودند. در نهایت مقادیر حساسیت، اختصاصیت، دقت، نمره F1 و صحت مدل به ترتیب برابر با ۸۵٪، ۹۸٪، ۸۶٪، ۸۴٪ و ۸۵٪ بود.

واژه‌های کلیدی: صحت مدل، ماتریس درهم ریختگی، مدل EfficientNet

۱- گروه مهندسی تولید و ژنتیک گیاهی، دانشکده کشاورزی، آب، غذا و فراسودمندها، واحد اصفهان (خوراسگان)، دانشگاه آزاد اسلامی، اصفهان، ایران
(*) - نویسنده مسئول: Email: imanahmadi1358@iaau.ac.ir