# Grapevine Disease Detection using Convolutional Neural Networks

Authors: Laith Darras, Timofey Lisensky

## Abstract

As discussed during lecture, the uses for machine learning systems or “AI” as they are often called right now span many different industries such as automotive, agricultural, and robotics. Despite the varied number of real world applications that exist for AI, the core system being used in most of these contexts is some variation of an object detection and classification model that takes in visual data and quantifies what it is seeing, such as differentiating between different things or counting how many of the same thing are in frame. In this experiment, we explore the uses of machine learning in industrial agriculture by training an image classification model that can differentiate between healthy grapevine leaves and leaves infected with a fungal disease called esca or grapevine measles with a convolutional neural network trained using Google’s TensorFlow library.

## Relevance

Rapid spread of crop blight is a natural result of human agriculture and is nearly unavoidable when growing lots of homogeneous plants adjacent to one another. The scale of modern industrial agriculture makes it impractical to manually inspect every plant for diseases, allowing blights to spread unnoticed until they reach a point where entire crops can be destroyed. The most common proactive approaches include using fungicide or choosing resistant plant varieties, however using a classification model to analyze a vast amount of plant data and only inspect plants that our model flagged for human review would save immense time.

## Dataset

The dataset we used was a pre-labeled set of nearly 2000 pictures of grapevine leaves found re-posted on Kaggle but originally created as part of a collaborative project between an Italian university and winery. Half of the images are of healthy leaves, and the other half show signs of being infected by the esca disease.

## Training

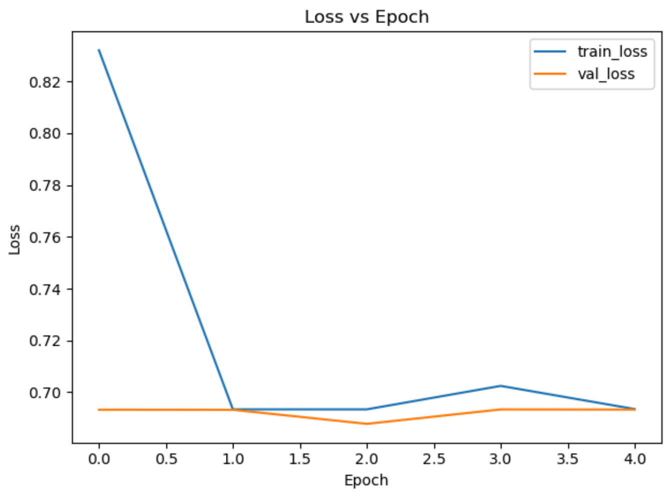
The TensorFlow library abstracts away most of the math in training a machine learning model, but allows us to control the most important parameters such as batch size and number of generations, as well as conveniently showing the model's improvement with a training loss value every generation. This shows us how well the model's predictions match up with the actual information in the training dataset, but does not necessarily show us whether or not the end result can accurately predict the validation and test data.

In our particular case, in order to train the model in a reasonable amount of time on a laptop we used a batch size of 16 and reduced the quality of each image down to 128x128 pixels, meaning that in every iteration of the model's training (of which there are five) it will process 16 of these small images at a time until it goes through the entire training dataset, then evaluates the results of that epoch using a portion of the set reserved for that purpose and tweaks some parameters.

## Evaluation

To evaluate our results, 100 images were set aside before training or validation and then fed into the finished model to compute the loss between the final result and these "new" images that had not been used as input during the training process as well as a couple of other values used for figuring out the false positive and false negative rate. This gave some interesting results, as the model seems to heavily prefer labeling an image as "infected with esca" over "healthy".

Our training loss started off around 0.85 after the first epoch and the greatest change was clearly seen from between the transition between the first and second epochs. From there it jumped around the same value for the next four generations and ended at 0.697. The validation loss however, stayed at around the same value the whole time. This is disappointing, as it suggests that while the model "learned" to predict the values in the training dataset it did not get any better at classifying images it was not directly trained on.

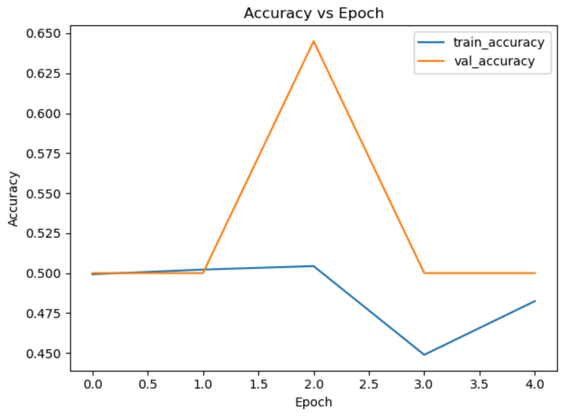


The precision of the model when classifying esca is 0.498, which means that the ratio of correctly classified true positives to all classified positives in the dataset is 0.498 as is shown by this equation.

$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP}$$

This means that about half of our images classified as leaves with esca do not actually have it and are healthy, so the false positive rate is quite high. Depending on the context a model is being used in it can be a bad thing, but in our case and in our case means that the model is not really saving us any time. Since there are only two categories, esca and healthy, this means that the model classified almost all of the plants as having esca. Generally a high rate of false positives is okay in this kind of application because the cost of false negatives is much greater, when the model just classifies everything as diseased it isn't saving any time because a human will have to review the same amount of photos as they usually would.

The accuracy plot, which shows the rate of accurate predictions whether they are esca or not reflects this finding as well. Both lines on the plot fluctuate, but settle where they started out which suggests that the model is in theory better than random chance but only because it marks everything as diseased, forcing humans to review all cases.



The recall of our model is 0.99, meaning 99 out of 100 plants with esca will be correctly recognized. This is a great result and is worth the tradeoff of low precision in theory because it means that the chance of a diseased plant going uncaught and spreading the disease is only one out of every one hundred predictions. Below is the equation for calculating recall, which is the same as the equation for precision except for the false positive rate being swapped with false negatives to represent all positive results in the dataset.

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

## Observations

The trained model achieved very high recall but comparatively low precision, indicating a strong bias toward predicting the diseased class. This depicts that the model tends to classify many leaves as infected, resulting in a high number of false positives. While such behavior may be acceptable in contexts where missing a diseased plant is costly, it limits the model’s practical usefulness due to reduced precision. Likely contributing factors include the limited dataset size, visual similarity between healthy and diseased leaves, and the absence of class-balancing techniques. Future work could focus on improving precision by incorporating class-balancing techniques during training, such as over-sampling methods or class-weighted training, as prior studies have shown that learning with limited examples in the presence of class overlap can bias model predictions. Applying these techniques may help reduce false positives and produce more well-defined results.

## Conclusion

In conclusion, the potential of using convolutional neural networks to prevent the spread of plant disease is promising and is already being used in industry. Our model, however, is clearly not accurate enough for real world use yet as the precision suggests that a human will need to review every single result anyway.

# References

Alessandrini, Michele; Calero Fuentes Rivera, Romel ; Falaschetti, Laura; Pau, Danilo; Tomaselli, Valeria; Turchetti, Claudio (2021), “ESCA-dataset”, Mendeley Data, V1, doi: 10.17632/89cnxc58kj.1

Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, *6*(1), 20–29. https://doi.org/10.1145/1007730.1007735

Google. (n.d.). *Classification: Accuracy, recall, precision, and related metrics | machine learning | google for developers*. Google. https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

*Image classification : Tensorflow Core*. TensorFlow. (n.d.). https://www.TensorFlow.org/tutorials/images/classification