Multi-class classification of healthy and infected leaves of crops plants

Computer Engineering for Robotics and Smart Industry

Academic year: 2021/2022

Authors

Emanuele Feola	VR474837
Davide Tonin	VR480503

Index

Motivation and rationale

“Human society needs to increase food production by an estimated 70% by 2050 to feed an expected population size that is predicted to be over 9 billion people. Currently, infectious diseases reduce the potential yield by an average of 40% with many farmers in the developing world experiencing yield losses as high as 100%.” [1].

Plant diseases play an important role in agriculture because they are very natural and failure to care will have serious consequences for plants and therefore affect the quality, quantity, or productivity of the product.

Timely and accurate diagnosis of leaf diseases plays a major part in preventing loss in productivity and loss or reduction of agricultural products. Detection of plant diseases by automated techniques is beneficial because it reduces monitoring efforts on large plants and detects an indication of disease that occurs on the leaves of plants very early.

Timely and accurate diagnosis of plant diseases is of great importance for sustainable and correct agriculture, as well as for preventing unnecessary waste of financial and other resources. Some plant diseases do not have visible symptoms and it is inevitable to use advanced analysis methods in such diseases.

“The automation of plant disease control is essential for early-stage symptom detection and continuous monitoring of crops. Such automation has a high impact on improving efficiency and productivity, especially in large fields” [2].

We were interested in multi-classification problems that could be useful to improve productivity and quality control in an industrial environment, and we think that this scenario may be a good exercise to develop and practice with AI techniques that are scalable also to other environments and use cases.

We particularly like this dataset because it’s well done, with a large number of images to train and evaluate the models.

State of the Art

After an overview of the previously published works on crop, fruit, and leaf disease classification, we noticed that the most used methods to solve the classification problem were based on convolutional neural networks and deep learning methods.
The principal solutions we found use CNN networks like AlexNet, VGG, ResNet and the more recent EfficientNet, with very accurate results leading to 95% accuracy levels in different papers.
By looking at the dataset images we decided to make a comparison between machine learning techniques (SVM, KNN) and neural network based solutions, to see if a neural network is really worth using.

Objectives

The goal of this project is to test Machine Learning techniques (SVM, KNN) and Convolutional Neural Networks (CNNs) on the PlantVillage dataset to perform classification of the images, and compare the obtained results.
In addition, using the Grad-CAM algorithm we show what area of the images are used by the Convolutional Neural Networks to make the classification prediction.

Methodology

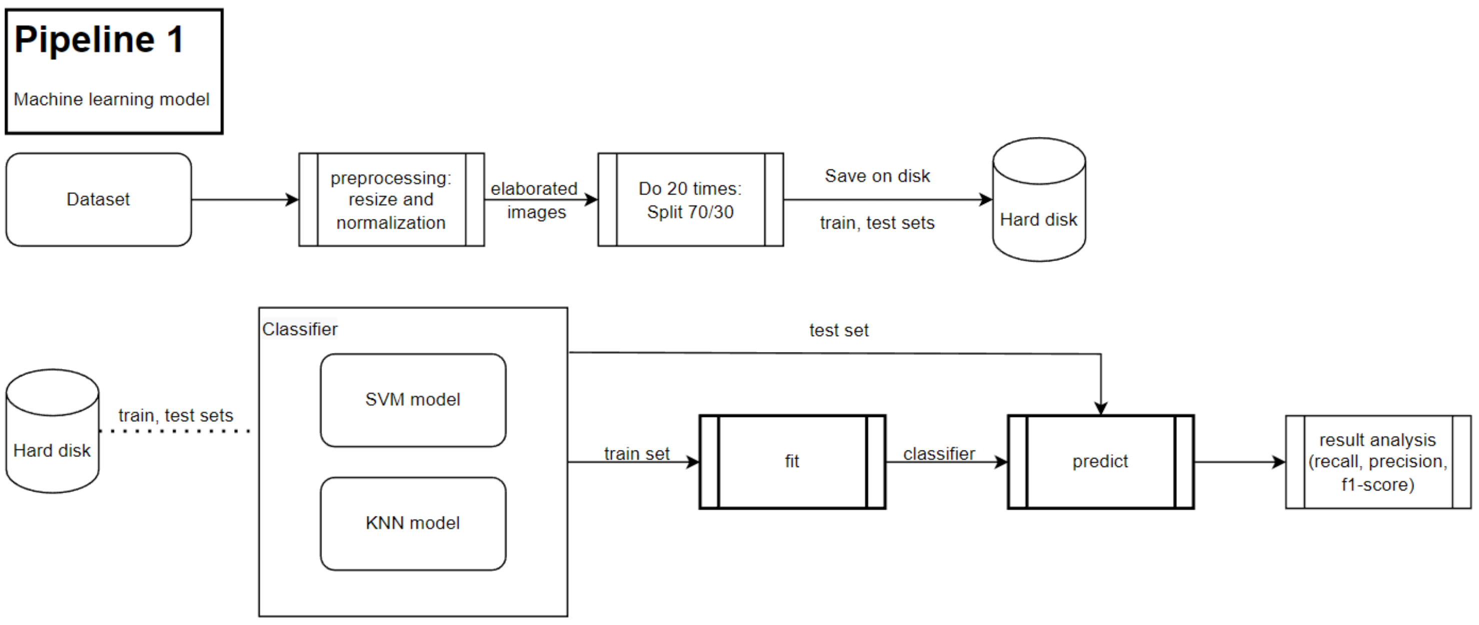
Methods, techniques and algorithms

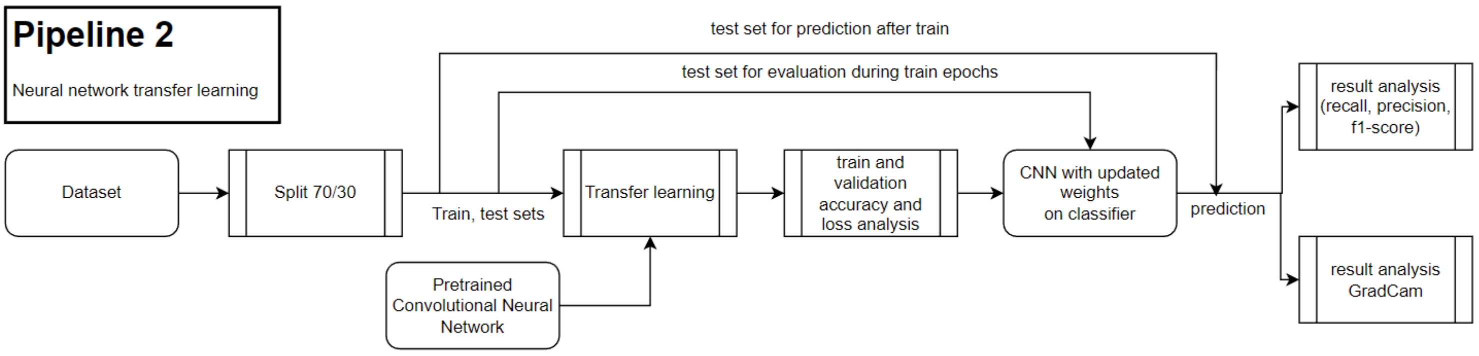
- Image preprocessing
- Resizing to fixed size and normalization
- Principal Component Analysis (PCA)
- Support Vector Machine (SVM)
- K-nearest neighbors (KNN)
- Convolutional neural network (CNN)
- Transfer learning
- Grad-CAM

Used libraries

- Scikit-learn
- Pytorch
- Opencv
- Numpy
- Matplotlib
- pyyaml

Pipeline




Pipeline 2
Neural network transfer learning
Dataset
Split 70/30
Train, test sets
test set for prediction after train
test set for evaluation during train epochs
Transfer learning
Pretrained Convolutional Neural Network
train and validation accuracy and loss analysis
CNN with updated weights on classifier
prediction
result analysis (recall, precision, f1-score)
result analysis GradCam

Dataset

[The dataset can be found on kaggle]

It consists of 15 classes and 20,639 images of 3 different plant leaves: pepper, potato and tomato.
12 of the 15 classes represent diseased leaves. The images in the dataset are RGB images of different sizes.
Examples of different tomato plant leafs in the dataset:

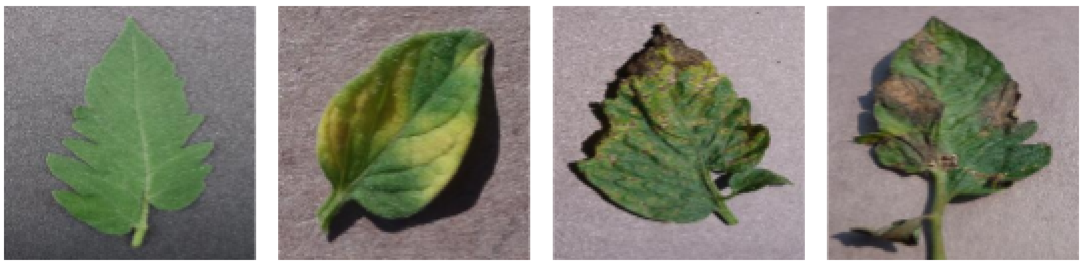
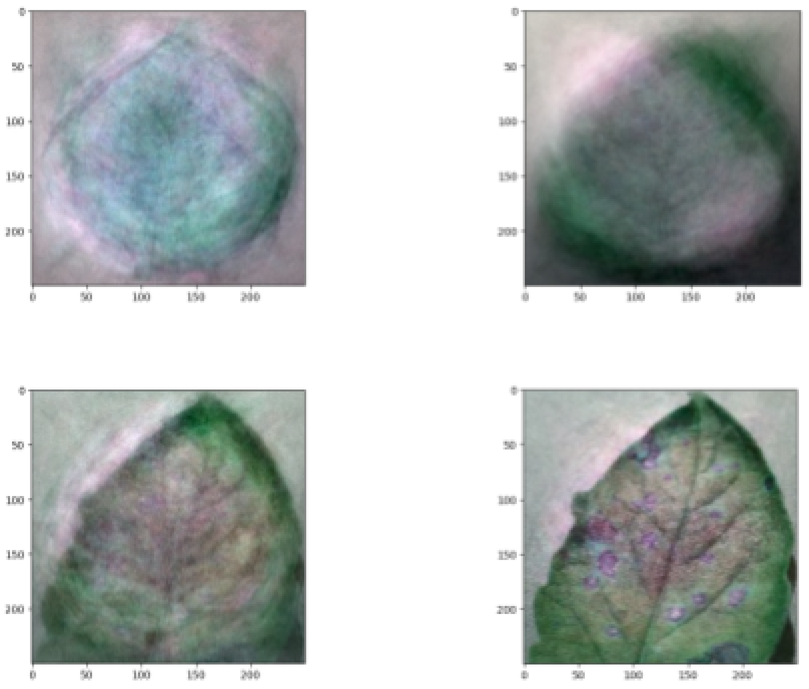


Image preprocessing operations

- **resize**: all the images are resized to the same dimension
- **scaling/normalization**
  - For the machine learning pipeline, we used sklearn's StandardScaler to standardize the features by removing the mean and scaling to unit variance.
    - the mean and the variance were computed over the training set, and then used to scale the test set
    - We also used sklearn’s MinMaxScaler but we didn’t notice any substantial difference
  - For the neural network pipeline, we used the pytorch Normalize with mean and standard deviation defined by the ImageNet dataset
- **cropping**: since the image resizing operation does not take into account the image proportion, we considered and tested cropping the images to a squared proportion before applying the resizing process

Dimensionality reduction - Principal Component Analysis (PCA)

In order to reduce the feature space, we used the PCA algorithm. We chose an 80% threshold, i.e. we kept the components that together consisted of 80% of the variance. To ensure the right number of components, we called the PCA constructor with n_components=0.8.
At first we implemented PCA from scratch like we did in lab lectures, but the computation was really slow, due to the number of images and their size.
So, we went with sklearn's built-in method which is much faster.
Here they follow 4 examples of the reconstructed images after the reduction operation at different variance percentages: 10%, 50%, 80%, 95%.



Experiments and results

Experiments and validation metrics

From the original dataset we built a training set and a testing set (with a 70/30 split).
For the machine learning pipeline (PCA, SVM and KNN), we created different train-test sets.
To ensure a degree of randomness in the creation of the train-test split, we changed the random seed every 3 iterations, for a total number of 9 iterations.
When a new algorithm/parameter had to be tested we executed the fitting procedure on the training set and ran a prediction on the testing set.
At the end of the 9 iterations on the same parameters set we computed the mean, std, and relative standard deviation of the results using the following validation metrics:
Validation metrics

- Accuracy
- Precision
- Recall
- F-score
- Confusion matrix

The above validation metrics were used to assess the quality of the classification output.
Each of the above metrics comes with its mean and standard deviation (and relative standard deviation - RSC), since the algorithm output was evaluated on a repeated number of executions.

For the neural network pipeline, we created a single train-test split using pytorch library to automatically load the dataset and the classes associated.

SVM

To properly test the SVM performance on the multi class classification problem we fit the the model to the data playing with different parameters:

- Kernel: rbf, linear, poly
    - Degree of the polynomial: 3, 5
- C: 0.1, 1, 10
- Gamma: scale

KNN

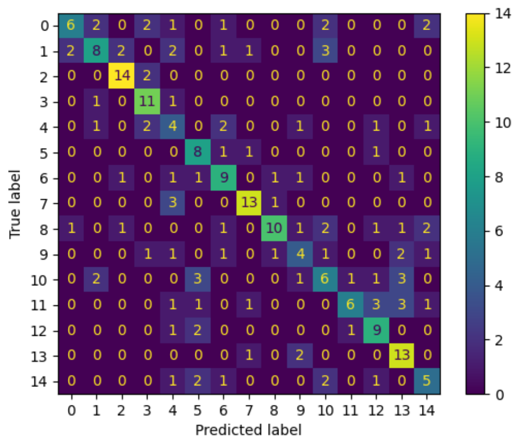
To properly test the KNN performance on the multi class classification problem we fit the the model to the data playing with different parameters:

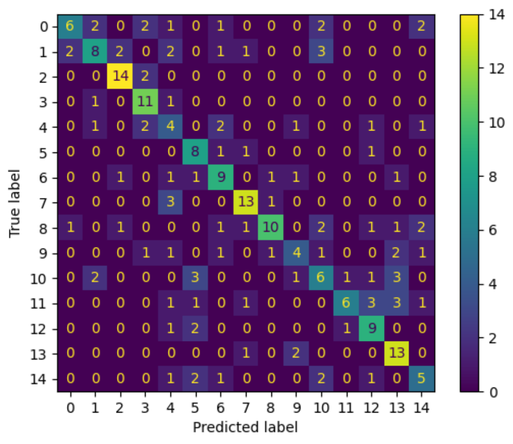
- Number of neighbors: 5, 11, 13, 25, 51
- Metric: euclidean, manhattan

SVM results

Train set size: 525 images
Test set size: 225 images

# tests	kernel	C	gamma	degree	Accuracy	Precision	Recall	F1-score	# mean support vectors for every class
9	linear	1	scale		0.49 +- 0.01 std (2.22% RSC)	0.50 +- 0.01 std (1.68% RSC)	0.50 +- 0.01 std (2.62% RSC)	0.48 +- 0.01 std (1.53% RSC)	[33.6 35. 33.6 34.3 35.3 35.6 34.3 28. 32.6 32. 30.6 33. 35.3 32.3 33.]
9	linear	10	scale		0.48 +- 0.02 std (3.26% RSC)	0.49 +- 0.01 std (2.35% RSC)	0.49 +- 0.02 std (3.72% RSC)	0.48 +- 0.01 std (2.34% RSC)	[33.6 35. 33.6 34.3 35.3 35.6 34.3 28. 32.6 32. 30.6 33. 35.3 32.3 33.]
9	rbf	1	scale		0.51 +- 0.02 std (4.35% RSC)	0.52 +- 0.02 std (4.04% RSC)	0.51 +- 0.03 std (5.64% RSC)	0.50 +- 0.02 std (4.78% RSC)	36. 34.8 34.4 33.6 35. 35.2 36.2 31.8 34.6 34.6 33.4 34.2 34.8 35.8 38.
9	rbf	10	scale		0.57 +- 0.01 std (1.29% RSC)	0.57 +- 0.01 std (2.61% RSC)	0.57 +- 0.01 std (1.79% RSC)	0.56 +- 0.01 std (2.42% RSC)	34. 35.6 36.6 35.3 35.6 35.6 35. 29. 35. 34.3 31.6 34.6 35.3 34. 38.
9	poly	0,1		3	0.13 +- 0.04 std (35.25% RSC)	0.12 +- 0.05 std (46.59% RSC)	0.15 +- 0.06 std (40.05% RSC)	0.09 +- 0.04 std (39.88% RSC)	34. 35.6 36.3 35.3 35.6 36.3 35. 32.6 35.3 34.6 31.6 34.6 35.3 34. 38.
9	poly	1		3	0.37 +- 0.05 std (13.76% RSC)	0.38 +- 0.06 std (16.38% RSC)	0.40 +- 0.04 std (9.26% RSC)	0.35 +- 0.06 std (15.82% RSC)	34. 35.6 34.3 35. 35.6 36. 35. 32.6 35.3 34.6 31.6 34.6 35.3 34. 38.
9	poly	10		3	0.32 +- 0.04 std (12.26% RSC)	0.38 +- 0.03 std (9.18% RSC)	0.35 +- 0.03 std (7.65% RSC)	0.30 +- 0.03 std (11.04% RSC)	34. 35.6 35. 34.6 35.6 35.6 34.3 32. 35.3 34.6 31.6 34.6 35.3 34. 36.3
9	poly	10		5	0.43 +- 0.02 std (4.44% RSC)	0.44 +- 0.03 std (6.30% RSC)	0.45 +- 0.01 std (3.15% RSC)	0.42 +- 0.02 std (5.38% RSC)	34. 35.6 34. 33. 35.6 35. 34. 31. 34.3 34. 31.6 33.6 35. 34. 35.



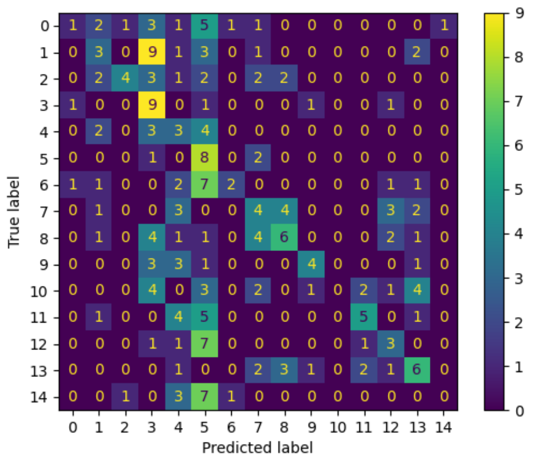


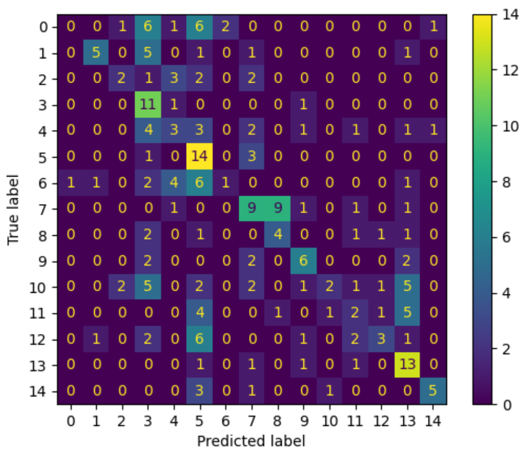
KNN results

Train set size: 525 images

Test set size: 225 images

# tests	# neighbors	metric	Accuracy	Precision	Recall	F1-score
9	11	euclidean	0.29 +- 0.05 std (16.43% RSC)	0.31 +- 0.06 std (19.19% RSC)	0.30 +- 0.05 std (16.15% RSC)	0.25 +- 0.05 std (20.22% RSC)
9	11	manhattan	0.21 +- 0.05 std (26.23% RSC)	0.29 +- 0.12 std (40.33% RSC)	0.21 +- 0.05 std (22.78% RSC)	0.16 +- 0.05 std (32.90% RSC)
9	5	euclidean	0.31 +- 0.04 std (13.46% RSC)	0.35 +- 0.04 std (10.77% RSC)	0.32 +- 0.04 std (11.42% RSC)	0.29 +- 0.04 std (14.38% RSC)
9	5	manhattan	0.24 +- 0.05 std (19.56% RSC)	0.31 +- 0.05 std (17.31% RSC)	0.24 +- 0.04 std (18.11% RSC)	0.20 +- 0.04 std (20.89% RSC)
9	13	euclidean	0.31 +- 0.05 std (16.25% RSC)	0.35 +- 0.04 std (12.70% RSC)	0.32 +- 0.05 std (14.68% RSC)	0.28 +- 0.05 std (19.36% RSC)
9	25	euclidean	0.27 +- 0.02 std (7.80% RSC)	0.33 +- 0.06 std (18.29% RSC)	0.29 +- 0.02 std (8.54% RSC)	0.23 +- 0.03 std (14.84% RSC)
9	51	euclidean	0.25 +- 0.02 std (8.95% RSC)	0.29 +- 0.01 std (4.46% RSC)	0.27 +- 0.03 std (9.39% RSC)	0.19 +- 0.03 std (16.08% RSC)





<u>SVM and KNN considerations</u>

By looking at SVM and KNN results, we can see they did not perform well on the multi class classification task.
As for SVM, the number of support vectors that are created is too high with respect to the number of samples. This is an indication of a high probability of overfitting.
For example, we can look at the case in the first row in which the SVM was executed with the following parameters: kernel=linear, C=1 and gamma=scale.
In this case the number of support vectors is about 33 for every class, resulting in 33*15=495 support vectors.
Since the sample size consists of 525 images, there are too many support vectors.

<u>SVM and KNN alternative classification problem</u>

One possible alternative to improve the SVM and KNN results, is to reduce the number of classes.
For example, one possible case scenario would be to reduce the problem to a binary classification one, in which we have to recognize if a leaf is either healthy or not-healthy.
By not classifying the type of leaf disease, we obtain better results.
The results shown below have higher validation results, with respect to the multi class classification problem.

SVM binary classification (healthy - not healthy) results

# tests	kernel	C	gamma	Accuracy	Precision	Recall	F1-score	# mean support vectors
100	linear	10	scale	0.81 +- 0.03 std (3.19% RSC)	0.83 +- 0.04 std (4.42% RSC)	0.75 +- 0.03 std (4.37% RSC)	0.77 +- 0.03 std (4.30% RSC)	14.3 20.6
100	rbf	10	scale	0.83 +- 0.05 std (5.79% RSC)	0.83 +- 0.05 std (6.55% RSC)	0.79 +- 0.06 std (7.82% RSC)	0.80 +- 0.06 std (7.33% RSC)	29. 50.3

KNN binary classification (healthy - not healthy) results

# tests	# neighbors	metric	Accuracy	Precision	Recall	F1-score
100	10	euclidean	0.64 +- 0.02 std (2.82% RSC)	0.68 +- 0.02 std (2.95% RSC)	0.69 +- 0.01 std (0.86% RSC)	0.64 +- 0.02 std (2.90% RSC)
100	5	euclidean	0.73 +- 0.01 std (1.44% RSC)	0.70 +- 0.03 std (3.83% RSC)	0.70 +- 0.03 std (4.87% RSC)	0.70 +- 0.03 std (4.28% RSC)

Convolutional Neural Networks and Transfer Learning

Since the results with machine learning algorithms like SVM and KNN provided poor results in the multi classification task, we decided to try some convolutional neural networks (CNN) models pre trained on the ImageNet dataset.
After a quick literature search, we saw that the most used CNN architectures are the following ones:

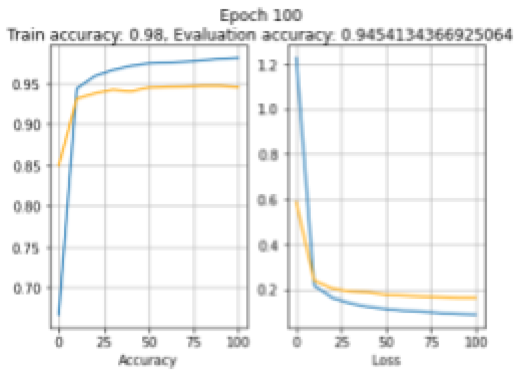
- Efficient Net
- Vgg
- ResNet

So we decided to train a CNN for image classification using transfer learning on our dataset.
We tried:

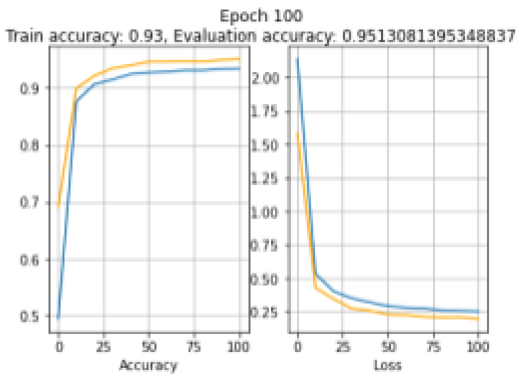
- Efficient Net B0
- Efficient Net B1
- Resnet 50

We freezed all the weights of the pre-trained CNN and we changed the structure of the last classifier layer, so that the output consists of 15 classes, that is the number of classes in our dataset.
By doing so, we kept unchanged the weights of the convolutional layers, which are already trained to extract features from the images.

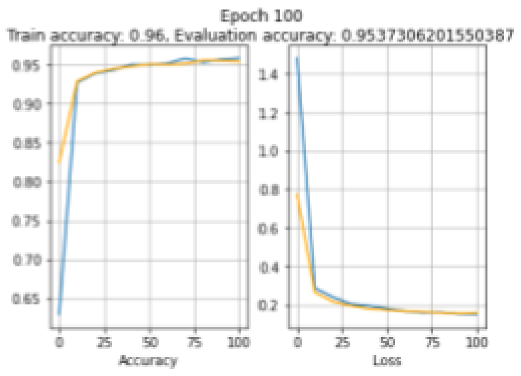
We then trained the classifier on our dataset using a 100 epochs period and a batch size of 250 images, since the train set consists of about 15000 images. Every 10 epochs we validated the model on the validation set and computed the accuracy and loss. The validation loss and validation accuracy were compared with those of the train set, in order to estimate the outcome performances and to recognize if overfitting was happening during the training phase.



Resnet 50



Efficient Net B1



Efficient Net B0

CNN parameters

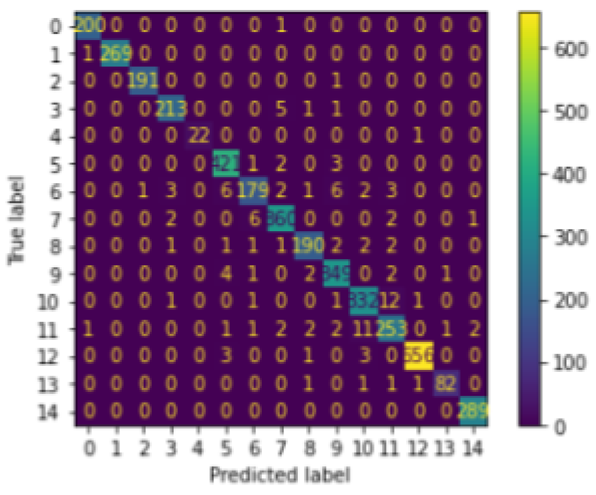
Before arriving to the shown results, we played a bit with a few parameters:

- Learning rate
- Momentum
- Optimizer (Adam and SGD)
- Pytorch stepLR scheduler parameters to introduce a decaying learning rate

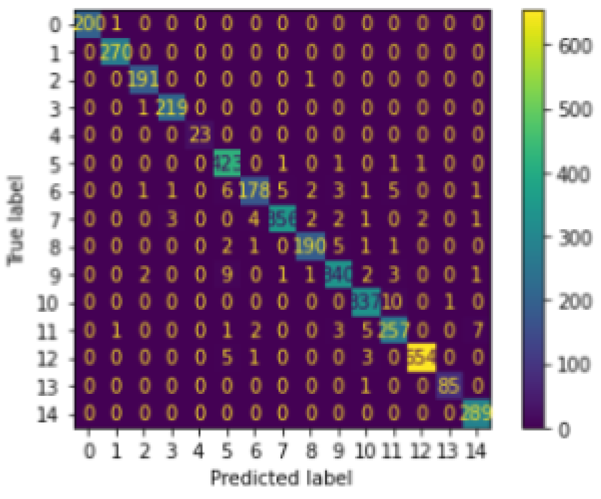
CNN results

Once the training phase was concluded we evaluated the trained model on the test set (about 5000 images) and gathered the following results that are grouped by CNN architecture.

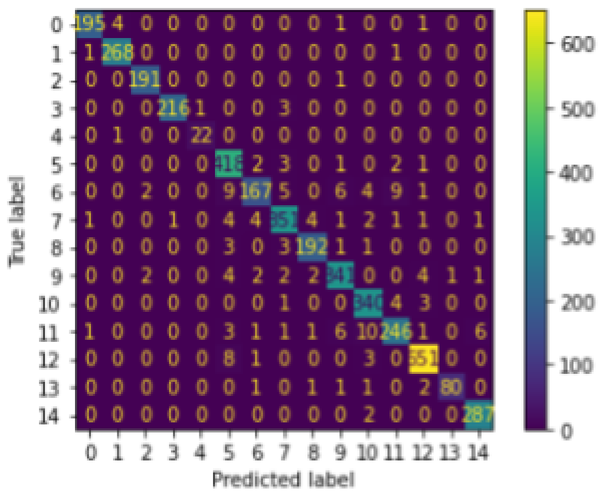
CNN base architecture	accuracy	precision	recall	f1 score
resnet50	0,9704	0,9712	0,9656	0,9683
efficientnet_b0	0,9719	0,9739	0,9722	0,9729
efficientnet_b1	0,9605	0,9626	0,9552	0,9585



Resnet 50



Efficient Net B0



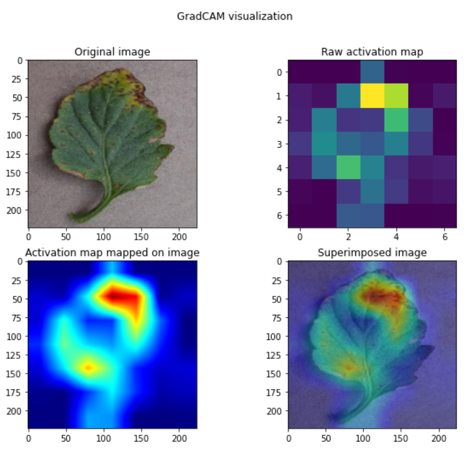
Efficient Net B1

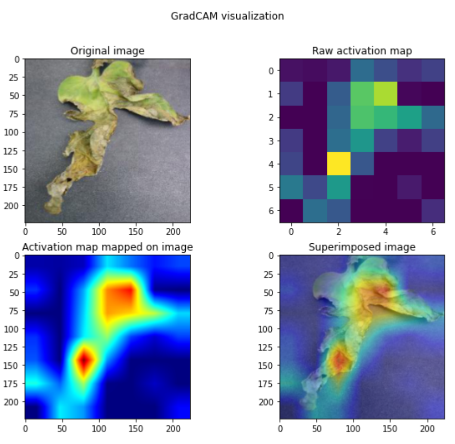
Grad-CAM - Interpretability matters

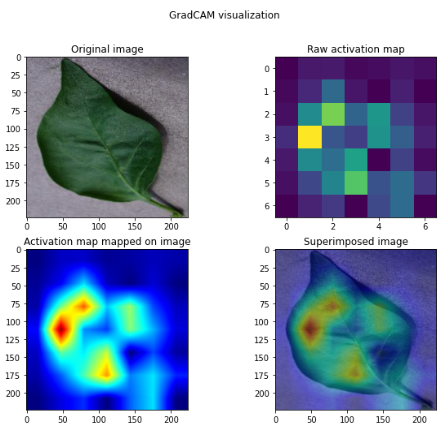
“Grad-CAM uses the gradients of any target concept flowing into the final convolutional layer to produce a coarse localization map highlighting the import regions in the image for predicting the concept”.[3]

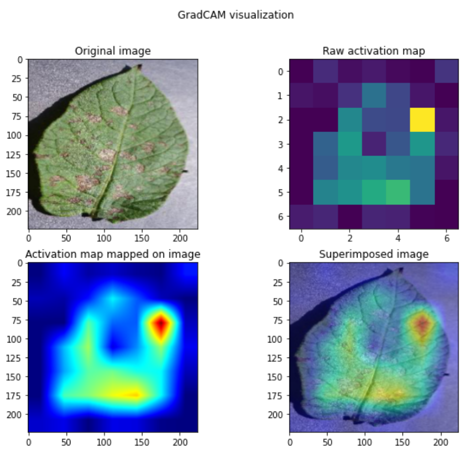
We were interested in finding out what are the parts in the images that are used to make the predictions, so we looked at the Gradient-weight Class Activation Mapping (Grad-CAM) algorithm, and we applied it to the trained model and a few test images.
By applying this algorithm, we can highlight and show the image areas that are used to predict the class.

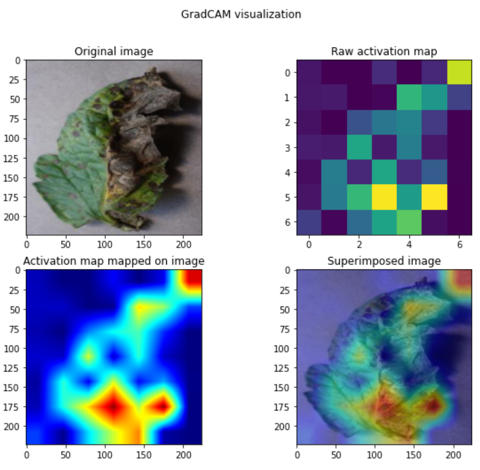
Grad-CAM examples

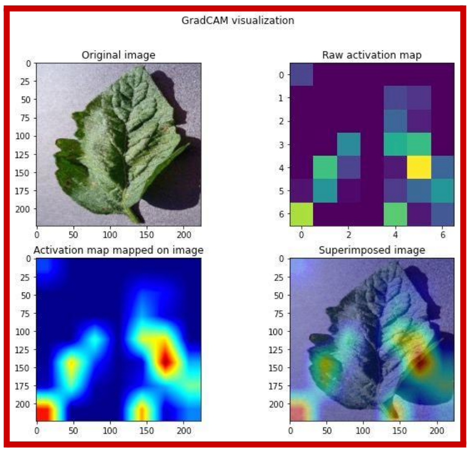












In the last example, the prediction was wrong and we can see by the activation map that the network doesn’t recognize the leaf so well.

Conclusions

The purpose of our project was to investigate if the neural network approach was needed to solve this multi class classification problem in which the objective is to recognize not only if the leaf is affected by a disease but also to classify the disease itself.
Our results show that the neural approach is better than SVM and KNN, especially due to the high number of classes in the dataset.

Future works

Future works may consist in developing the following steps:

- Image segmentation to identify the area on the leaf on which the disease is present
- Try some data augmentation techniques to increase the variability
- Try other SoA models

Bibliography - References

[1] *An open access repository of images on plant health to enable the development of mobile disease diagnostics*
[2] *Reliable Deep Learning Plant Leaf Disease Classification Based on Light-Chroma Separated Branches*
[3] Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization

Other links:
Improving the classification of invasive plant species by using continuous wavelet analysis and feature reduction techniques
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Github Code

The code can be found at this repository on Github