*The Impact of Filter Size and Number of Filters on Classification Accuracy in CNN [1]*
Starting from a predefined CNN with fixed 3 x 3 filter size and number of filters, we have modified these parameters to determine their impact on accuracy. This has been done through a series of experiments: in each one we have demonstrated that the accuracy varied according to filter sizes and number of filters that we were using. In order to obtain better generalization, the candidate filter sizes were {3 x 3, 5 x 5, 7 x 7}. The candidate number of filters were {32, 64, 128} to be applied, respectively, to 3 convolutional layers. In our first experiment results showed that when the filter size was fixed to 5 x 5 pixels and the number of filters were 32, 64 and 128, respectively on 3 convolutional layers, the accuracy was almost unvaried.Though, in a second experiment we achieved better performance with a 7 x 7 fixed filter size while maintaining the same architecture as before. Finally, we tried to combine different filter sizes 7 x 7, 5 x 5, 3 x 3 in a telescopical way on our model, still achieving similar performance as in the 7 x 7 case.

*The Impact of Convolutional layers on CNNs complexity and accuracy*
Taking notice of previous results, we have experimented on the numbers of convolutional layers in order to understand which level of complexity the model could reach by assessing overfitting.Starting from 3 convolutional layers with ReLu activation function and Stride=1 each one stack on a MaxPooling layer, we have increased the total number of convolutional layers to four by adding an additional convolutional layer with 3 x 3 filter size and 128 filters, obtaining the best performance. Then, we increased again the number of layers to 6 but in this case the model became too complex for the classification problem at hand, worsening the overall performance. Additionally, we tried to increase the Stride by 1 and verified, as shown in the literature, that on each experiment the accuracy worsened due to less precision in convolving the inputs.

*Choice of the best optimizer and weight initialization strategies*
Here, we have tried to assess whether, although Adam converges faster, the SGD generalizes better with greater performance.[2] Using SGD results in better performance with respect to Adam; another method, NAG, shows better performance on the previous models than SGD and Adam. It is also interesting to note how SGD and NAG optimizers introduce fluctuations in the final graph with respect to Adam. We can reduce fluctuations by adding momentum.
The result of gradient descent depends on the initialization of weights, so it is important to choose appropriate weights to increase network performance and achieve efficient backpropagation. There are several options for initializing weights, but we decided to base our choice on Xavier and He's techniques, due to their proven effect in providing adequate and distributed learning, employing less time to reach convergence and demonstrating to be a valid choice for deep architectures (these three problems are typical of more traditional initialization methods). [3]

*Pooling Layers (overall number and pooling techniques)*
Pooling layers help to make the representation approximately invariant to small translations of the inputs. We tested common pooling functions: average pooling and max pooling. They calculate the average and maximum value, respectively, for each patch on the feature map. [4] We kept only max pooling because it performed better in our tests. The GAP layer calculates the average output of each feature map in the previous layer and reduces the dimensions to a 1-dimensional vector. Dimensionality reduction decreases the chances of overfitting. [5] In our tests, the use of GAP instead of Flatten considerably increased the accuracy on the test set.

*Activation functions, hidden layers, and hidden units*

We researched the most common activation functions: Sigmoid, Tanh, Relu, and Leaky Relu. Then, we tested Relu and Leaky Relu. Relu offers better performance than sigmoid and tanh since it does not suffer from the vanishing gradient problem; however, it does suffer from the dying neuron problem. Leaky Relu does not suffer from the dying neuron problem although it has the vanishing gradient problem for negative values of the function. [6] In the end we kept only Relu as the performances were similar.
For many problems a hidden layer is sufficient. Using two hidden layers rarely improves the model and increases the risk of converging to local minima. [7] We decided to test our network with one and two hidden layers. Because the results were slightly better, we chose a network with two hidden layers.
To select the number of hidden layers, in the absence of an exact rule, we used the rule of thumb: "The number of hidden neurons should be between the size of the input layer and the size of the output layer." [7]

*Training*

We chose the cross-entropy loss function since it is widely used for multi-class classification problems. [8] To improve the algorithm's performance, we added two callbacks to the fit method: Early Stopping and Reduce Learning Rate on Plateau. Early stopping reduces the risk of overfitting by stopping training if there are no improvements on the validation set. [9] Reduce learning rate on Plateau allows for better tuning when the algorithm is close to the best result, avoiding learning from stalling.

*Transfer Learning Overview in CNNs*

The natural next step to try gaining better results for our problem was taking advantage of transfer learning, which is a popular technique used in Machine Learning when training neural networks in order to save computational power and improve results, when dealing with relatively small datasets. The general idea is to take a model developed for one task and re-use it on a second related task. The pre-trained model is trained on a larger dataset in order to learn general features that can be used to improve the results of the related task using a smaller dataset. The two models must share the same inputs otherwise pre-processing should be applied. In most cases the pre-trained model has more neurons in the final output layer than the ones we require, in such scenarios, we need to change the final output layer accordingly (in our case 8 output classes).

*Explored Options Overview*

To take advantage of Transfer Learning, we tested the following pre-trained models:
*VGG-16, VGG-19,ResNet-152, and Inception V3.* We further enhanced our model by combining fine tuning techniques with our transfer learning experiments. Fine-tuning involves unfreezing parts of the pre-trained model and training the entire model once again on the whole dataset.

*Final Results*

VGG-16 obtained an overall accuracy comparable with the one obtained by VGG-19 but not satisfactory. Thus, we experimented on ResNet which resulted in the worst performance. The best performance was achieved by the Inception V3 architecture with an accuracy of ~80%. In order to obtain this result, we added a Global Average Pooling layer to regularize the output and improve overfitting other than managing the computational load, taking notice of its positive effect when introduced in our CNN model.

*Data Augmentation Overview in Convolutional Neural Networks*

Given data scarcity in our problem, having a total of 2709 training samples after splitting (very little against deep learning standards), we decided to implement Data augmentation techniques to extend the inter-variance of our training set and expand the overall number of samples available. To gain a general understanding of the possible transformation usually applied in deep learning applications we explored several options, including all the basic geometric and photometric manipulations commonly employed. [10]

Several possibilities were evaluated, simulating them on samples of images taken from the dataset. We evaluated all the transformation techniques reported in the reference paper.

Among those, *shearing*, *color space shifting* and *blurring* were excluded.

With ImageDataGenerator we applied simple and multi-step transformations of the images, producing an augmented dataset with 5x overall numerosity.

Testing the dataset with different CNN architectures, we had discouraging results, with great separation from validation, suggesting strong overfitting: our hypothesis was of possible synthetic features fabricated by our manipulations that were not present in the validation and testing set. This probably happened due to the non-invariant nature of the transformation we decided to use.

Trying to find a more robust strategy, we search through studies dealing mainly with plant-based datasets. An interesting proposal came out of Zhang (2015), which used a simpler transformation with a stochastic approach to classify leaves.

We decided to emulate Zhang's transformation using preprocessing layers from keras and the preprocessing_function option of ImageDataGenerator.

The detailed description of the transformation can be found here.[11]

While Zhang's strategy looked less detrimental in terms of introducing synthetic features (in most architecture we had no drop or improvement on both accuracy and overfitting), we still found that data augmentation was not useful to improve our final result, and thus decided to discard it. We only registered a slight improvement of confusion in the unbalanced classes we had (1,6), as we tried to balance the dataset with augmentation due to the impossibility of downsampling (too few data). Still, this could not justify the adoption of the strategy, as there was no improvement (or even a worsening) on accuracy measures.

References:

*[1] https://doi.org/10.1007/s10115-022-01707-3*
*[2] https://doi.org/10.48550/arXiv.1509.01240*
*[3] https://doi.org/10.1007/s10462-021-10033-z*
*[4] Bengio, Y. (2016). Page 342,Deep Learning. MIT Press.*
*[5] https://doi.org/10.1007/978-3-030-71187-0_16*
*[6] https://doi.org/10.48550/arXiv.2004.06632*
*[7] http://dx.doi.org/10.7763/IJCTE.2011.V3.328 Pg. 332-337.*
*[8] https://doi.org/10.1016/j.neucom.2021.08.106*
*[9] https://doi.org/10.1007/s00365-006-0663-2*
*[10] https://doi.org/10.1007/s10462-021-10066-4*
*[11] https://doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.318*