Introduction

After a data exploration phase dataset was found to be unbalanced, and deemed as small, a total of 3542 images are available for 8 classes. Some images have blurry shaded areas. The first thing that was done was consider weighted classes. Data was split with a .85-.15 training validation ratio respectively. After a basic mean variance analysis by class, it was found that variance was distributed evenly among a considerable image area, no focal areas with high variance were found for any class, this observed on file ANNDL%20PROJECT%201%204.ipynb.

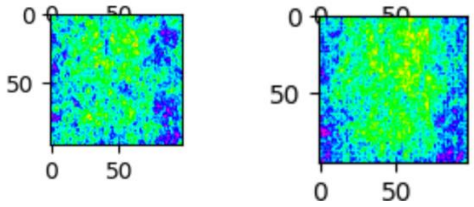

Figure 1. Mean and Variance for class 1, dataset

Data augmentation

After the findings that came out of data exploration data augmentation data included in all proposed models. Rotations, shift, flip, and zoom were performed on the images to obtain a higher number of training samples. We paid attention to select transformations that were not affecting the characteristics of the different species (color, form, etc.) because that's the information of interest to train models over, this also after doing data exploration analysis on all classes. At the same time, a non-augmented dataset was considered.

Use of homemade models

An intuitive step done was to propose own models entirely from scratch. After some trials with a few stacked layers models, it was noticed a deeper network was needed because validation rates obtained were under 50%, at best, and with tendencies to overfitting after few epochs. We agreed on using popular architectures, giving up with this solution, because well-known architectures are empirically proved to extract features effectively and generalize.

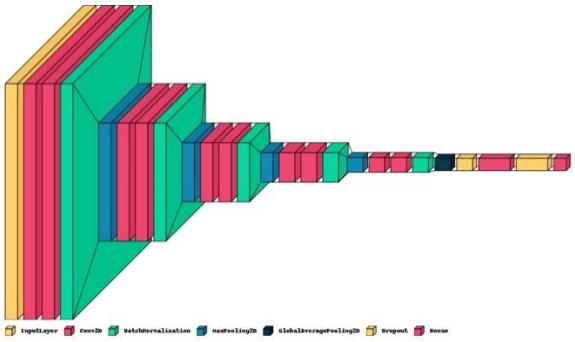

Figure 2. Proposed Neural Networks

Transfer learning

From the previous experience it was decided to resort to better suited architectures this because such architectures have solved more complex problems, are well documented both in literature and forums; however, after conducting experiments the following scenarios were observed.

- VGGs: The model used the most, both VGG16 and VGG19, were leading to better generalizations with moderate computing times.
- EfficientNet, results marginally lower than VGG, with similar computational cost.
- Resnet, was considered, given the fact the first two listed were promisingly performing, this one was found to perform 5% less that EfficientNet with erratic accuracy values observed over epochs.
- The following models performance was not close to the ones from previous models: **inception, Xception.**

Fine-Tuning

Fine Tuning is good solution in case of small dataset, as it reduces the number of layers (and so the number of parameters) to train. We consequently tried to apply fine tuning considering the VGG16 network that was giving us the best results applying transfer learning. It allowed to obtain around 0.71 on the hidden testing set.

After that, multiple stage fine tuning was applied. For this, we considered the model trained using Transfer Learning, the VGG16 architecture and data augmentation. We froze some layers and trained it without the data augmentation, and we repeated the same process (transfer learning with data augmentation and then fine-tuning without) multiple times until no further improvement was observed on accuracy. Our goal in doing so was to improve the capability of the network to generalize.

Ensemble Model

We also tried to assemble models. We trained three models (VVG16, EfficientNet, ResNet) using the transfer learning method, obtaining all accuracy around 70%. We then decided to include them in only one model, averaging the output to have more stable results. Doing so didn't really improve the results directly performing the prediction. However, training the assemble model to make the sub models working together it increases the accuracy on the testing set up to 79%

We also tried to implement it with models that were performing better than those three, but in most cases, it decreases accuracy as this is a more rigid method and it cannot face unknown samples.

The last try was about ensemble similar model architecture (VGG16 and VGG19) that were trained using the multiple stage fine-tuning. In total, we used four different models with different configuration: vgg16 trained with transfer learning (65% accuracy), vgg16 with 2 fine-tuned layers (81%), vgg16 with 6 fine-tuned layers (85%), vgg16 with all layers fine-tuned (89%), vgg19 with 9 layers fine-tuned (0.9%) and vgg199 with all layers fine-tuned (92%).

This last model on the hidden test set reaches a 87% accuracy, while the ensemble reaches a 89% accuracy. All models were computed using the vgg16 preprocessing on data, as both functions are compatible.

Preprocessing

A standardize methodology was discussed and explored when homemade models were deemed as an option, adding a grayscale image as a fourth channel, low pass filtering images, as computer vison

methodologies suggest, on the grayscale image Sobel filtering was applied to highlight sharp transitions, information hidden on the blurred shades revealing sharp transitions hidden by blurring. Applying this preprocessing pipeline led to no substantial improvement on homemade model, poor generalization experiments so it was finally considered to only blur a 3 channel images, model2.py contain a function called exp1, applying such pipeline was not considered necessary after observing the good generalization potential of VGG16 built-in preprocessing function and VGG16 architecture. All data was preprocessed with that function for simplicity and better generalization.

Parameters and classifier delivered

The first layer dense layer, output of the VGG16 architecture was with 512 unit with using 512 images batch. We tried to complexify the classifier by increasing the number of dense layers, but it did not improve generalization only increased computing time. In this case, we tried to also reduce the overfitting by increasing the dropout, but it often taking a toll on final performance. Besides, we tried also to tune the learning rate and the batch size, but no significant difference was noticed, other than the speed to which model was trained. All the steps taken to create the model that was delivered can be found on ToSubmitCode.ipynb.

Conclusion

The best network we built and the one we propose to solve this problem is consequently the one using both multiple stage fine-tuning and ensemble of different models. We acknowledge however that the efficiency and time complexity to train it was probably too high and that the use of the best vgg19 multiple stage fine-tuned model could have been a better and simpler solution, acceptable results tend to be obtained with proven architectures, otherwise they won' be trendy.

One further improvement we could have investigate is to consider the possibility to combine classes (the one with less samples) so to consider a two-stage classifier first determining among the group of class, and then selecting the class. Creating a stage classifier, that makes the problem easier to generalize, but with the fact that doing web search not much was found this idea was early discarded. With more time, we could also have implemented a convolutional neural network able to unblurry images to get better quality images by manually blurring unblurred images, label them and then train it to reconstruct blurred images, instead of using preprocessing pipelines, the labeling and blurring itself would have took a considerable amount of time, that given the results leaded by multi stage fine tuning investing such effort was also early discarded.