ANNDL Challenge 1 Report

Alessio Facincani ⊗ Mousa Sondoqah ⊗ Mohanad Diab

Looking at the data

For the first approach, it is always best to look first at the data that we will have to work on. The dataset is composed of 8 classes of various foliage, so it's a classification problem, very suitable for a Deep Learning approach. The classes have roughly the same amount of data each, with the exception of classes 1 and 6 of which we're provided fewer data. The photos are taken from above, in different lighting and shadow conditions, the leaves can be in any direction and amount inside the frame. We tried many different approaches and network configurations to address the challenge, and we will hereby describe the main line of thinking - although not so linear - with successes and failures.

Making a network

The very first try was to build and train a network from scratch, exploiting the widely used stacking composed of multiple convolution+pooling layerings with a final Fully Connected layer.
However, this benchmark approach was not of any success in terms of accuracy, since it yielded poor results with regard to the methods used afterwards (roughly 70% accuracy on local test sets).

Transfer learning

At first we were trying various network architectures available in the keras API, but then we decided to let the cloud computing service do all the work and wrote a script (`CNN_variations.ipynb`) that imports all the models available on keras applications API and then builds a model using the CNN part of the imported networks, sticking a softmax fully connected layer with 8 outputs (matching the number of the classes), which will be a benchmark performance test to get insights on which models perform the best on just 3 epochs.
The keras API provides 28 models to export, and testing the dataset on each one has not been a walk in the park, but the output of the script was very insightful to our further work, based on the performance output of the models. It turned out that, for our specific problem, and based on how this test was run, the "DenseNet201" was the best performing model.
Thus, following the previous test, our testing space has been reduced since now we have "frozen" the CNN part in our search for the best model, from this point most of our work has been solely focused on variations of fully connected layers stacked above the DenseNet201 CNN (Top not included) and variations of data augmentation. In this way we are restricting our search space to converge to a better model by fixing some parts of the model design process. Figure (1) in the appendix shows an overview of the pipeline adapted for this challenge.

Dataset imbalance

As mentioned before, some classes in the given dataset experienced clear imbalance, especially for classes 1 & 6. Such imbalance means that the model will have less training samples within these two classes. One of the approaches we used to deal with this issue was to use the `class_weight` parameter in the `model.fit` function, the weight for each class is calculated based on the number of samples it has, so the fewer images the class has the larger the weight for the class. Such wights adjustment will give more attention to the classes with fewer number of images.

Augmentation

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of the images in the dataset. It's nothing but a set of techniques to increase the diversity of your training set by applying random - but realistic - transformations, such as rotation. Such techniques can improve the way the deep learning model learns since it offers more variations of the training set to be seen by the model.
As such, it is clear that the choice of the specific data augmentation techniques used for a training dataset must be chosen carefully and within the context of the training dataset and knowledge of the problem domain. After an extensive investigation of the provided training set, we concluded that a reasonable choice of augmentation parameters could be as following:

```
horizontal_flip=True, vertical_flip=True, brightness_range=[.2, 1.2]
```

Such configuration will always generate images similar to what could come from the original generating process. We noticed that applying transformations such as rotation and zoom would decrease the overall accuracy, likely due to the artificial fill that the generator has to apply.

Keras Tuner

For the hyperparameter optimization we used the `keras-tuner` package (script `CNN_tuning.ipynb`) . We already had a good idea of an architecture that would work for the fully connected part, since we tried different stuff prior to this step. We only needed the tuner to search the space of parameters to possibly reduce some of the overfitting problems we were facing.

Figure (2) in the appendix shows the final architecture used and the parameters that were found by the tuner, along with their limits set while tuning. The keras tuner also tunes the learning rate based on all the previous inputs, and is designed to augment based on the validation loss using Bayesian Optimization. The tuner runs for 5 epochs on 30 different trials (models), each run converges to the optimum solution based on the Bayesian approach. The final suggested best model was very specific, so we fixed the final values to be 1024, 0.5, 512, 0.5 for the Dense, dropout, dense, dropout from left to right respectively.

Final model

After searching for the best configuration of the DenseNet201, we had to perform the final training for the model, as shown in Figure (1): the first part being the transfer learning with the layers in the DenseNet201 CNN frozen, while the second is un-freezing the whole network and running the same model on a lower learning rate. Wrapping everything up, we submitted this model to CodaLab obtaining an accuracy of 87.4% in the first phase and 85.8% in the second phase. From the confusion matrix we could see that the detection of the first species was the hardest point for all the models, obtaining only 70% accuracy for that species. Species 6 was the other one with fewer data, but overall it gets recognized well, while Species 8 is another one that looks overall problematic. This shows that the amount of given data is not always a problem, probably the data provided for Species 6 was representative enough for its class.

Failed attempts

Image-data generation

One of the main characteristics of the provided image dataset in this challenge was the imbalance it had, especially for species 1 and 6 which have less samples than the other species. One of the approaches we tried to deal with this imbalance was trying to generate more samples by using augmentation. The difference between the usual augmentation - which is done for the data for training purposes - and the augmentation used here is that the latter is performed without replacement of the original images so the augmented images will be added to the original samples. However, training over such a dataset showed bad performance for the trained model, which could be explained by referring to the variations created by the augmentation and learnt by the model that do not necessarily exist in the testing set, so we settled to just use the `class_weight` parameter as explained previously, and apply a basic global augmentation.

Works on Color Space

Due to the nature of the problem, we also thought about a completely different approach: exploiting the different Color Spaces available for encoding an image. RGB is only one of them, there are many others that are proven to separate various aspects of the same image in each channel - but as stated in [3], the best color space to use depends on the particular application.

One of the first examples that came to mind is **HSV** (Hue, Saturation, Brightness): the idea for this challenge was to build a network that exploits the Hue channel to detect where the green part is (ideally ignoring the parts with e.g. red/brown terrain), skip the Saturation channel (to possibly being invariant to different lighting and exposure conditions), and finally use the Brightness channel (equivalent to grayscale) to detect the shape of leaves present in the portion where the Hue channel said they're present.
With more thorough research on color spaces, we found two interesting color spaces: **I1I2I3** proposed by MICC in [2] which is supposed to be invariant to highlight effects, and **C1C2C3** which is invariant to shadowing effects. These color spaces are often used in image segmentation, proving our intuition. A sample of our dataset converted to these color spaces is provided in Figure (3). The samples are chosen to show e.g. the effect of the Hue channel in HSV being able to locate the green portion of the image, or C1C2C3 channels being invariant to shadowing.

We coded the color space conversion functions for these and other color spaces, confident that with at least some of them the network would perform well, but unfortunately it was not the case. Results were not terrible, but they could not compete with our models working on the usual RGB images. One reason is probably that we were not able to effectively apply Transfer Learning to this different color approach, since all the famous networks work with RGB images and thus cannot apply well to the different learning approach entailed by the color space change - but, with more time and dedicated research, we believe it is possible to train a model from scratch which can perform well, while possibly being much smaller in size.

Appendix

Figures

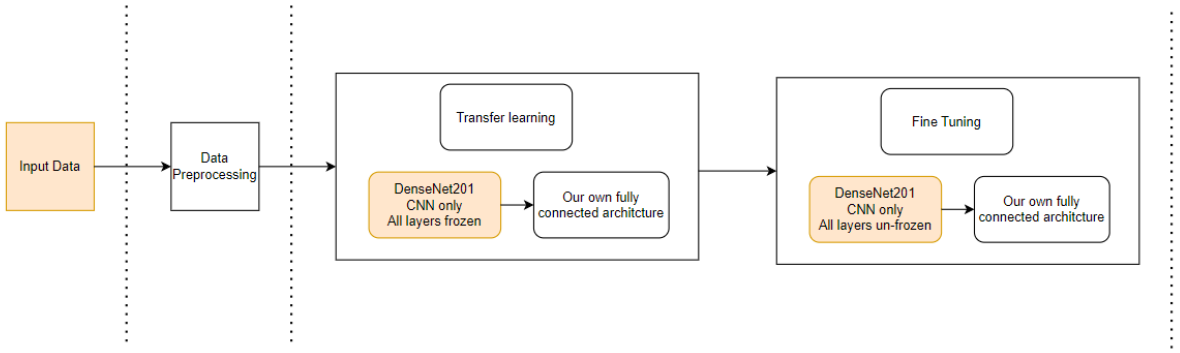

Figure 1: training pipeline overview, showing the parameters that are initially fixed (yellow) from those which are not and have to be discussed (white)

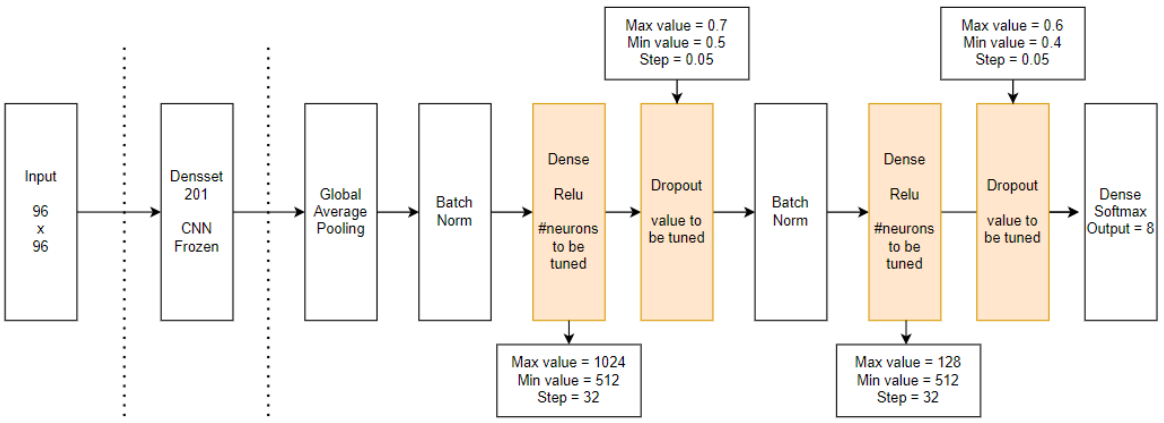

Figure 2: the proposed architecture of the fully connected parts following the CNN taken from DenseNet201 with hyperparameter optimization.

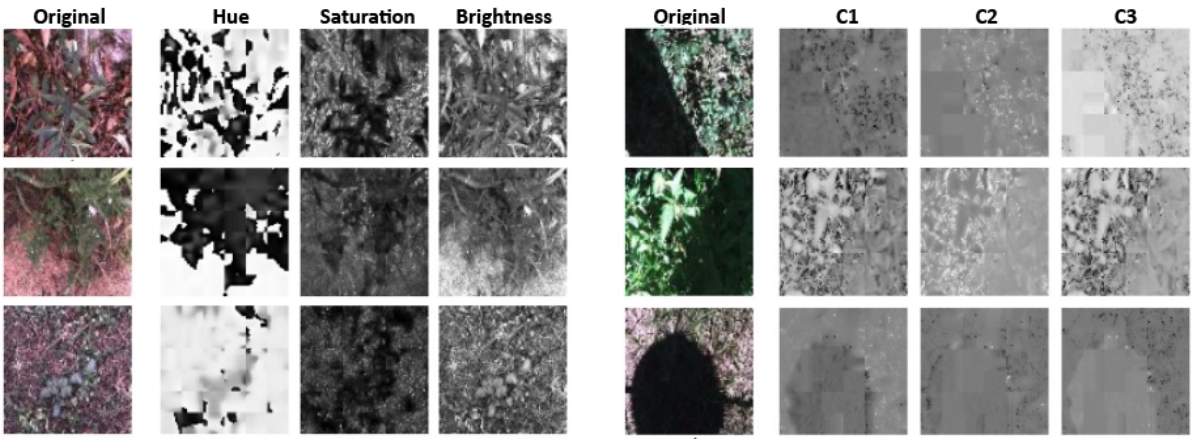

Figure 3: A few examples of our dataset converted to HSV (left) and C1C2C3 (right) color spaces.

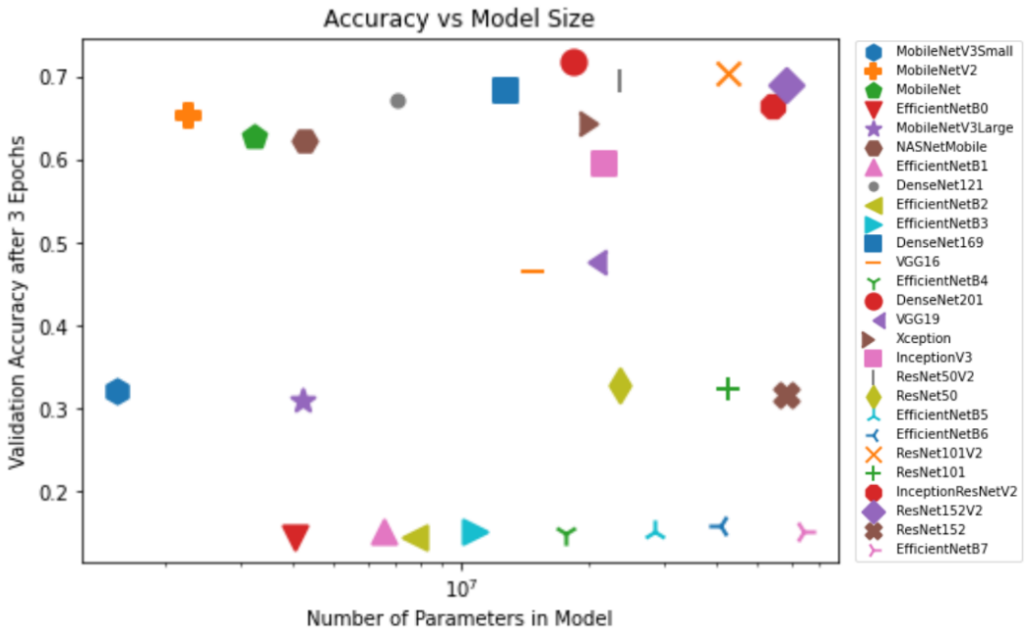

Figure 4: The accuracy of the listed models vs # of parameters after 3 epochs

	model_name	num_model_params	validation_accuracy
16	MobileNetV3Small	1529968	0.321530
14	MobileNetV2	2257984	0.654391
13	MobileNet	3228864	0.627479
3	EfficientNetB0	4049571	0.145892
15	MobileNetV3Large	4226432	0.310198
17	NASNetMobile	4269716	0.623229
4	EfficientNetB1	6575239	0.151558
0	DenseNet121	7037504	0.672805
5	EfficientNetB2	7768569	0.144476
6	EfficientNetB3	10783535	0.151558
1	DenseNet169	12642880	0.684136
24	VGG16	14714688	0.464589
7	EfficientNetB4	17673823	0.150142
2	DenseNet201	18321984	0.718130
25	VGG19	20024384	0.475921
26	Xception	20861480	0.644476
12	InceptionV3	21802784	0.594901
23	ResNet50V2	23564800	0.695467
22	ResNet50	23587712	0.328612
8	EfficientNetB5	28513527	0.152975
9	EfficientNetB6	40960143	0.160057
19	ResNet101V2	42626560	0.705382
18	ResNet101	42658176	0.324363
11	InceptionResNetV2	54336736	0.665722

Figure 5: The numerical values of accuracy and number of parameters after 3 epochs of training

Figures list

Bibliography

[1] T. Gevers, A.W.M. Smeulders, *Color-based object recognition*, Pattern Recogn. 32 (1999) pp. 453–464. https://staff.fnwi.uva.nl/th.gevers/pub/GeversPR99.pdf

[2] Elena Salvador, Andrea Cavallaro, Touradj Ebrahimi, *Cast shadow segmentation using invariant color features*, Computer Vision and Image Understanding 95 (2004), pp. 238–259. https://doi.org/10.1016/j.cviu.2004.03.008

[3] Rasouli, A., & Tsotsos, J. K. (2017). *The Effect of Color Space Selection on Detectability and Discriminability of Colored Objects*. arXiv. https://doi.org/10.48550/arXiv.1702.05421