Examination of the Plantnet-300K data set with optimization options and ensemble methods

[redacted]

[redacted]

Studiengang: Master Data Science

[redacted]

Table of Contents

Table of Figures

Introduction

This projects examines and works with the PlantNet-300K dataset. The PlantNet-300k dataset is a publicly available image dataset. The code used is based on the GitHub repository https://github.com/plantnet/PlantNet-300K that is also used to produce the Paper *Pl@ntNet-300K: a plant image dataset with high label ambiguity and a long-tailed distribution* (Garcin, et al. 2021).

In addition to reproducing the model from the GitHub repository, an exploratory data analysis is performed and four additional options for training are implemented: Using data augmentation for underrepresented classes, using a weighted DataLoader based on the class distribution, using a weighted loss function based on the class distribution, and using an existing model as a checkpoint in combination with the aforementioned three options.

PlantNet also provides a variety of models trained on different architectures to classify image data. The examination ends with three approaches to combine the already trained models. The goal is to highlight how already existing models can be leveraged to increase the overall accuracy of a system. The first two approaches are ensemble methods where each model's prediction is either simply averaged in the first approach, or, in the second approach, additionally weighted based on the overall model's accuracy on the test set. The final part of this project is a Mixture of Experts approach where a router, also called a gating network, is used to dynamically determine the weights each model has given an input image. In line with the goal of reusing existing models, the models are not additionally trained, only an additional routing network is trained in this final part.

Exploratory Data Analysis (EDA)

In this chapter an Exploratory Data Analysis is performed to understand the dataset and the distribution within the dataset. As the dataset was provided and already split in train, validation and test sets, the first objective is to verify that the distribution within each class between the train, validation and test sets is about the same. As can be seen in Figure 1 the distribution between the different sets is identical, with a split of 80 percent of images used for training, 10 percent of the images used for validation, and 10 percent of the images used for testing.

class	count_test	count_train	count_val	total	train_rel	val_rel	test_rel
1363227	902	7208	902	9012	0.799822	0.100089	0.100089
1392475	792	6337	793	7922	0.799924	0.100101	0.099975
1356022	767	6140	768	7675	0.800000	0.100065	0.099935
1364099	668	5334	668	6670	0.799700	0.100150	0.100150
1355937	648	5178	648	6474	0.799815	0.100093	0.100093
...	...	...	...	...	...	...	...
1423932	1	2	1	4	0.500000	0.250000	0.250000
1423633	1	2	1	4	0.500000	0.250000	0.250000
1420700	1	2	1	4	0.500000	0.250000	0.250000
1360976	1	2	1	4	0.500000	0.250000	0.250000
1718287	1	2	1	4	0.500000	0.250000	0.250000

Figure 1: Distribution of the train, validation and test set

Next up the data distribution across classes has to be looked at. Figure 2 shows the data distribution across all classes. The class labels are not shown as there are 1.081 classes in the dataset. However the figure shows that the dataset is highly skewed. A very small minority of classes is responsible for the majority of the images. It is likely that models trained on this dataset won't be providing accurate results for the majority of the classes that only have a few images.

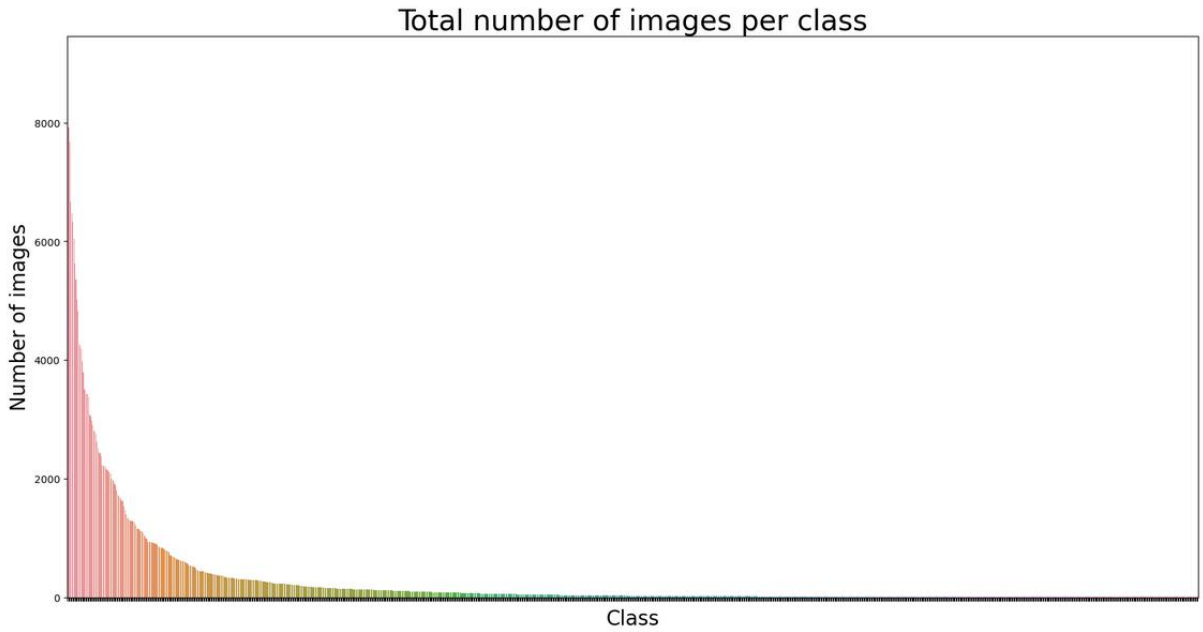

Figure 2: Distribution of the number of images per class

The skewed distribution is even clearer when grouping the classes in 10 percent percentiles. Figure 3 and Figure 4 show the results of this transformation. Only 3 classes are responsible for providing over 10 percent of the total dataset. Just 38 classes out of the 1,081 classes make up more than 50 percent of the dataset. The final 10 percent contains 850 classes.

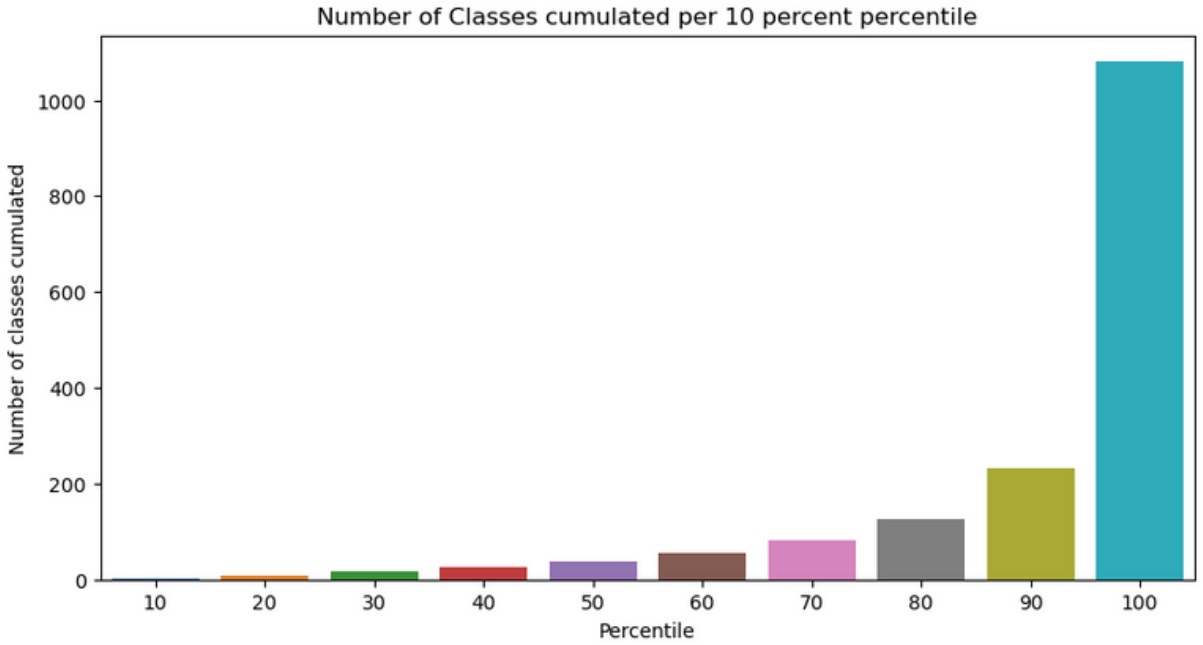

Figure 3: Number of classes per 10 percent percentile

percentile	number_of_classes
10	3
20	9
30	16
40	25
50	38
60	54
70	81
80	125
90	231
100	1081

Figure 4: Table number of classes per 10 percent percentile

As shown there is a large range of number of images across the different classes. Another interesting analysis for this dataset is the accuracy of a model per class sorted by the number of images for that class. The accuracy of a model in a class does not solely depend on the amount of data available for that class, but also on the quality of the data and how closely the training data and validation data correspond to the test data. The following numbers highlight the accuracy of a single model and are all based on the ResNet-18 model trained by PlantNet. In the later chapter *Data Augmentation for underrepresented classes* the model is retrained to validate the GitHub repository and the provided models.

As can be seen in Figure 5 a stable accuracy measurement can only be seen from around 1,000 images in a class and higher. Below that threshold the accuracy of the model per class appears to be more random.

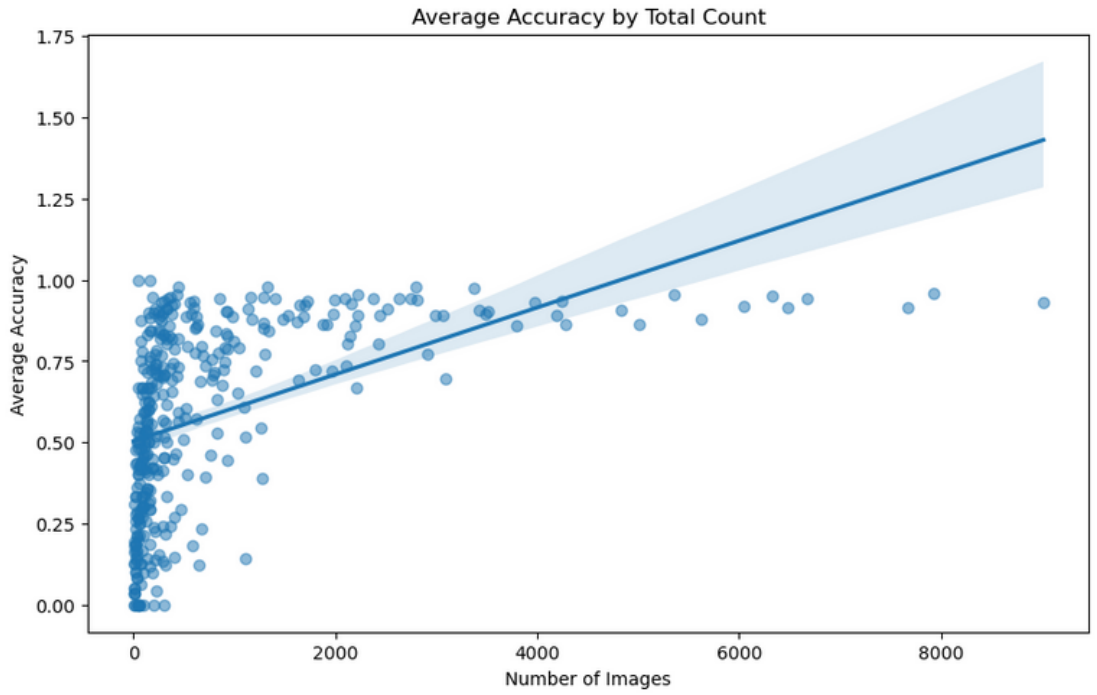

Figure 5: Average Accuracy by total image count

Another way of visualizing the point made in Figure 5 is shown in Figure 6. For this figure the accuracy percentage is floored to the closest integer and the number of images in each bin is averaged. This emphasizes the number of images that are on average needed for an accuracy measurement. It can be deduced from the result that for achieving an accuracy of over 90%, an accuracy that would be considered good, on average at least 1,000 images are needed. For context: The average accuracy for the ResNet-18 model was about 77.9%, and just 93 classes had an accuracy of over 90%.

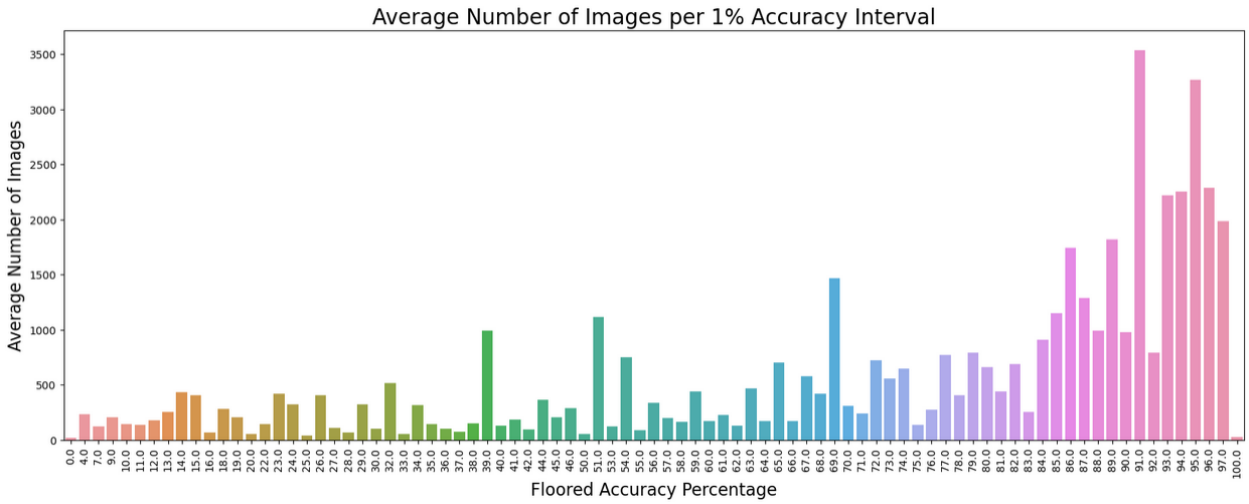

Figure 6: Average Number of Images per 1% Accuracy Interval

The last part that was looked at during this phase of the analysis is the training time. For training the model the in the GitHub Repository provided CLI call was used. Training the model for 30 epochs took about 3 ½ hours in optimal conditions where the server was not occupied with other calculations. During that time the GPUs were never fully utilized. Only one of the four available GPUs was used as the provided code was not parallelized and it was also not rewritten for parallelization within this project. The average GPU utilization for the used GPU during the training phase was about 30%, as shown in Figure 7.

```
nvidia-smi

Sat Apr 27 11:16:59 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.65.07    Driver Version: 515.65.07    CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  On   | 00000000:01:00.0 Off |                    0 |
| N/A   39C    P0   140W / 500W |   6237MiB / 81920MiB |     28%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-SXM...  On   | 00000000:41:00.0 Off |                    0 |
| N/A   34C    P0    63W / 500W |      0MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-SXM...  On   | 00000000:81:00.0 Off |                    0 |
| N/A   31C    P0    72W / 500W |      0MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A100-SXM...  On   | 00000000:C1:00.0 Off |                    0 |
| N/A   29C    P0    66W / 500W |      0MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Figure 7: GPU Training Utilization

Figure 8 displays the complete training history for the ResNet-18-model. A clear cut can be seen at epoch 20 as learning rate decay was used for epoch 20 and 25. The decay at epoch 25 does not show a significant impact. Learning rate decay means that at epoch 20 and 25 the learning rate was decreased, in this specific implementation to 10% of the previous value. This helps the model to finetune the weights more precisely once the model already optimized the weights with larger learning rates.

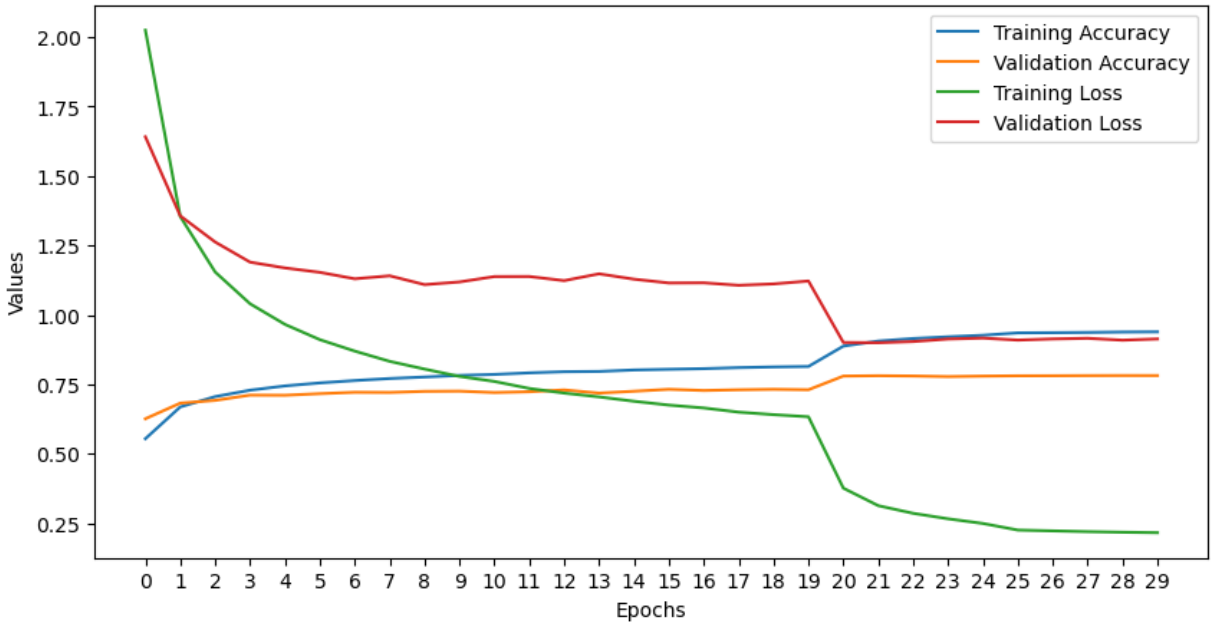

Figure 8: Training and Validation Accuracy per Epoch for the basic model

Optimization strategies for improving a single model

In this chapter four different approaches for improving the model accuracy are described. These approaches are data augmentation, oversampling and undersampling classes by changing the DataLoader, oversampling and undersampling classes by changing the loss function, and continued training of a model in conjunction with a combination of the three before mentioned ideas.

All results mentioned in the subchapters are based on the ResNet-18-architecture.

Data Augmentation for underrepresented classes

As shown before the dataset is highly skewed. One way to reduce the skewness of the model is to use data augmentation on only the underrepresented classes when training the model. The idea is that by augmenting the underrepresented images, meaning by rotating, flipping, changing the perspective, using a slight blur, or by using a slight colour change, the model sees a larger number of different images for the underrepresented classes.

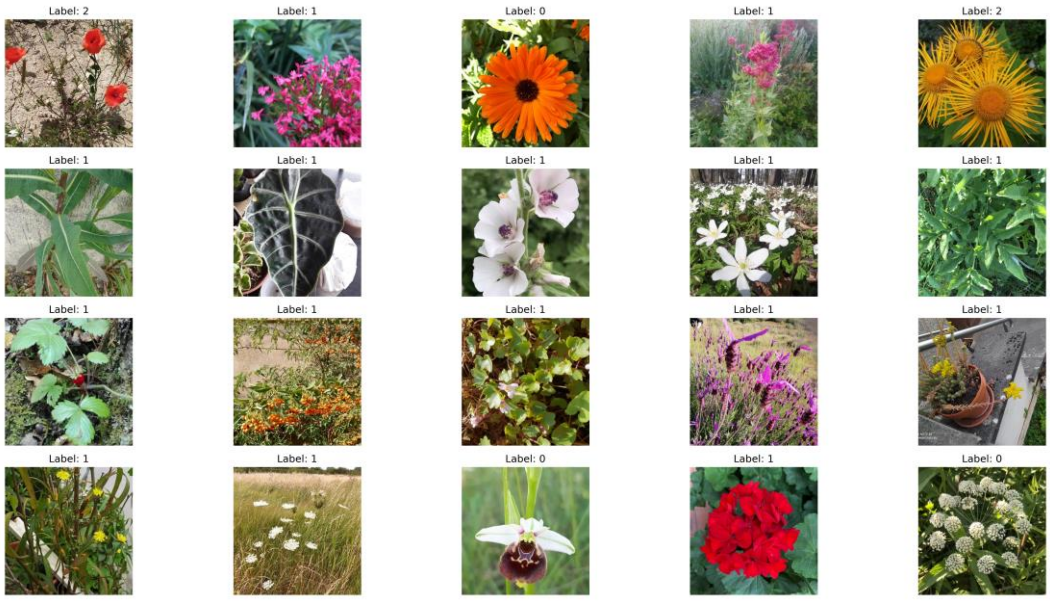

Figure 9: Non-Augmented images

Figure 9 shows 20 example images from the dataset. Figure 10 shows 20 example images with additional augmentation. The augmentation might seem small to the human eye, but image models tend to be quite sensitive to small changes. More aggressive augmentations were also tested but they lead to noticeably worse performing models. When using too aggressive augmentation the training data is no longer in line with the real world test data, which is why the model performance decreases quite rapidly when using too aggressive augmentation.

Figure 10: Augmented images

In the final settings used for data augmentation the overall accuracy on the test set was 77.8%, and 94 classes had an accuracy of over 90%. This is almost exactly the same level as the general ResNet-18 model (77.9% accuracy, 93 classes accuracy over 90%).

Weighting of the DataLoader

Another approach is oversampling of underrepresented classes and undersampling of overrepresented classes by directly changing the DataLoader. A probability is used to load the images from the dataset. In this implementation the probability is defined as the inverse of the number of images in a class.

$$class\_weight = \frac{1}{number\ of\ images}$$

This changes the distribution of the data the model sees during training. The model trains more on images of the underrepresented classes.

This approach decreased the accuracy of the model. The overall accuracy for this model on the test set is 69.1% with only 83 classes having an accuracy of over 90%.

Weighted loss function

The main idea of using a weighted loss function is similar to the approach mentioned before, but it is not identical. This approach also uses the *class_weight* based on the inverse of the number of images in a class, but instead of changing the input to the model directly, the loss function weighs the input. The model still uses the same distribution of data as in the standard process for training, but it weighs the training results differently based on the class an image belongs to.

This weighted loss approach performed even worse than the weighted sampling approach. The overall accuracy for this model on the test set is 50.8% with only 53 classes having an accuracy of over 90%.

Continued training of an existing model

The last change to the original implementation is that reading a model in as a checkpoint is now possible instead of only being able to retrain the models from scratch or using general pretrained weights based on foundational datasets like ImageNet. This approach can be combined with any or all of the three before mentioned additional methods.

This is more a functionality than an approach. The results depend on the additional parameters used. In the tested approach continued training was combined with data augmentation for classes that have less than 1,000 images. 20 epochs of additional training were performed. The overall accuracy decreased to 72.5%, but the number of classes with accuracy over 90% decreased dramatically to 47 classes, almost half of the original number (93 classes).

Comparing these numbers to the two weighted approaches mentioned before, as the overall performance of this model is over the accuracy of the weighted models, but less classes show an accuracy of over 90%, this model might detect the underrepresented classes better than the original model. Further analysis would be necessary to make this statement with confidence, but this additional analysis is not part of this project.

Ensemble Methods

All analysis and optimization strategies mentioned before focused on one specific model. A different approach to the problem of increasing accuracy is not based on one model, but on multiple models. These ensemble methods can be broadly split into two categories: Ensemble methods, which use every model in an ensemble for every input image, and a Mixture of Experts approach which utilizes an additional neural network, a router or sometimes called a gating network, to route the image either to the model that is most likely to give the correct answer, or to dynamically weigh the response of multiple models, depending on the input image, to calculate the final result.

For the ensembles 26 models were used. The overall accuracy of the models ranged from 68.7% for AlexNet up to 82.1% for the Vit-Base-Patch16-224 architecture.

Average and weighted Ensemble

The easiest and most intuitive way to create an ensemble of models is to get the result of every single model for an input image and averaging the results. The resulting ensemble performed better than the best single model at an average accuracy of 83.17%.

Another way of creating a still relatively simple ensemble is to use fixed weights for each models' result based on the average accuracy the individual model achieved on the test set. This approach outperformed the simple average ensemble approach slightly at an average accuracy of 83.20%.

Achieving a higher accuracy is great in theory, but in most real world scenarios compute and response time have to be taken into consideration. The downside of both ensemble methods is that every single input is fed into every single model, so the needed compute and response time increases with every model used in the ensemble. Where a single model had gone through the test set in about 40 to 45 seconds, the ensembles had to do that for every single one of the 26 used models, so the total test time was about 21 minutes and 30 seconds. An ensemble with this many different models might not be the right choice for user facing real world applications.

Router Mixture of Experts

A way to mitigate the problem of compute and response time is to use a router or gating network with the models. This way a router decides which input should be routed to which model to increase the overall accuracy of the model while still only requiring the compute and response time of one expert model, in addition to the compute and response time of the router. This approach is called Mixture of Experts.

In a classical Mixture of Experts approach the router and the experts are trained together during the training phase. The expert models are also trained on different subsets of the data to decrease their needed compute for training and to create actual *experts* on parts of the data. During training the router learns the general subsets of the data, routes the input further to that specific expert, and the expert calculates the final prediction.

The Mixture of Experts approach can also be used in the model architecture itself to replace fully connected layers in a transformer with Mixture of Expert layers (Fedus, Dean und Zoph 2022)**.** When

experts are trained on parts of the dataset, it is also possible to partition the dataset down further through multiple routing layers into a tree-like structure (Eigen, Ranzato und Sutskever 2013).

In this implementation the already trained models are used with a router. The implementation also does not route an input to the singular best expert model, but it uses the results of every single model and dynamically sets the weights for every single model based on the input.

There are pros and cons to using already trained models in a router compared to training the experts and router together. A pro to the implemented approach is that the individual models are already trained, so no additional compute other than training the router is needed. Additionally the models are independent models that can be trained fully asynchronously to every other model, which might make sense in some landscapes. A con to this approach is that the overall compute needed for training all individual models and the router, each on the whole dataset, is higher than training the router and experts for subsets of the data.

The implemented approach can be optimized via different additional methods. For example the current implementation does not use load-balancing. Load-balancing is the process of ensuring that each expert is used equally by the router, so that not just one single model does the majority of predictions. Based on the system landscape, where in production different models might be on different machines, this might help prevent bottlenecks. Load-balancing can be achieved by using auxiliary losses during training, or by treating it as a linear assignment problem (Fedus, Dean und Zoph 2022). There was also work done that omits the problem of load-balancing through an objective function completely by implementing fixed, non-learned routing patterns (Roller, et al. 2021).

The implementation also does not use a router architecture that is specialized on the dataset. Today the Mixture of Experts approach is mainly used in the area of Large Language Models (LLMs). The Mixture of Experts approach finds its way into image recognition tasks, but there are less clear best practices for implementing a router in the special use case of already trained models as experts. Work in image recognition tasks seems to focus on the Mixture of Experts as a layer approach (Riquelme, et al. 2021). Because of that in this implementation one of the existing general image recognition architectures is used and slightly altered. The router in this implementation is based on the ResNet-50 architecture. The main change is the replacement of the last layer by a softmax layer corresponding to the number of existing models in order not to determine an input image class directly, but to enable routing to the expert models.

The router also does not implement enough measures against overfitting. The router, even though in the current implementation it was limited to only 5 models to save on training time and compute, is overfitting, with a training accuracy of 99% and a validation accuracy of 76.7%. A dropout layer is implemented, but additional dropout layers or regularization methods should be used to decrease the degree of overfitting. Choosing a different neural network architecture for the router might also help in decreasing overfitting. The final test accuracy is 77.5%, which is lower than using the ResNet-50 architecture directly for classifying the images (test accuracy for ResNet-50 models: 80.7%). For comparing the router with a fair benchmark, both ensemble approaches were also calculated based on the same 5 models. The simply averaged ensemble has an accuracy of 82.1% and the weighted ensemble has an accuracy of 82.3%, so significantly better than the current router approach.

Overall the approach of reusing already trained models with a router seems promising, but a lot of work has to be done to get all the parts correct, so that the accuracy remains high and the model is not overfitting to the training data.

Conclusion

The ensemble methods of multiple existing models proved to be a viable solution for increasing the accuracy over every single model. However one also has to put this into the appropriate use case. The inference time for the ensemble method is dependent on the number of models used in the ensemble. It might not be suitable, especially for user-facing applications, to use 26 models in an ensemble as the inference time would increase too much compared to a single model. However, in creating a fair comparison for the Mixture of Experts approach, it was also shown that using just a few models in a weighted ensemble still outperformed every single individual model, even though the models were not handpicked to complement each other. The accuracy was slightly below the ensemble of all models, but this might be a viable option, trading accuracy for inference time and compute, depending on the use case.

Getting the already trained models together in a Mixture of Experts approach is more difficult. As the models are already trained, only the router had to be trained. However a specific router architecture has to be designed or additional measures for overfitting and load-balancing need to be implemented. The approach itself seems promising, but getting all the parts working correctly, defining a functional router that is not overfitting and utilizes load-balancing and other best practices needs time. In real world applications it might be a more viable option to combine a few already trained models in a relatively simple ensemble with fixed weights to enhance the accuracy over every single individual model, at least for the specific use case where already trained models exist.

References

Eigen, David, Marc Aurelio Ranzato, and Ilya Sutskever. *Learning Factored Representations in a Deep Mixture of Experts.* ArXiv, 2013.

Fedus, William, Jeff Dean, and Barret Zoph. *A Review of Sparse Expert Models in Deep Learning.* ArXiv, 2022.

Garcin, Camille, et al. *Pl@ntNet-300K: a plant image dataset with high label.* NeurIPS 2021 - 35th Conference on Neural Information Processing Systems, 2021.

Riquelme, Carlos , et al. *Scaling Vision with Sparse Mixture of Experts.* CoRR, 2021.

Roller, Stephen, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. *Hash Layers For Large Sparse Models.* CoRR, 2021.

Appendix

Jupyter Notebook plantnet_300k_ensemble_MoE: This file contains most of the code for this project.

Accuracy_plot.html: An interactive visualization of Figure 5 that shows the average accuracy by total image count.

Python files: Additional new code, especially for the optimization strategies, had to be made in the original files to train the additional models. All code parts I implemented in the files are commented by *start of my additional code* and *end of my additional code*. All files except *epoch.py* contain additional code.

- epoch.py
- main.py
- utils.py
- cli.py