Deep Learning: Plant Pathology 2020 - FGVC7

Bine Markelj

1 Introduction

In this report we present our complete solution to Plant Pathology 2020 - FGVC7. We decided to work on an project that was a part of a Kaggle challenge, called Plant Pathology 2020 - FGVC7 [1]. The goal was predicting potential diseases, based on images of apple leafs. We created the whole pipeline from data preprocessing to training and evaluating different models and ensembles of models. We compare performance of EfficientNet, ResNet and Vision Transformer models and their ensembles. We achieved comparable results to other competitors. In the report we fully outline the whole pipeline, our experiments, reasoning behind pipeline decisions and report the performance.

2 Experiment

2.1 Problem overview and dataset

The motivation for the problem comes from agricultural importance of apples for food. The effect of diseases on many such orchards could be very tragic for very vital food production. The current disease identification is being done by humans, however it is very expensive and time consuming. This is the main reason, that the process should be automatized using a trained model. Such model should be very robust and able to classify samples in different environments, of different illuminations, colors, etc.

The main task of the task is to train a successful model, that based on presented leaf image classifies the apple's health. There are 4 possible classes. This are: 'healthy', 'scab', 'multiple_diseases', 'rust'. Each leaf image can be classified into 1 class. Dataset consists of around 3642 high-quality RGB images of apple leafs. This dataset reflects real field scenarios by representing non-homogeneous backgrounds of leaf images taken at different maturity stages and at different times of day under different focal camera settings.

Half of the dataset is labeled, this will be used to train our classification models. We first analyzed the dataset and found that the distribution between classes is skewed at class 'multiple_diseases', which has much less samples. The dataset distribution is also shown on Figure 1.

The final evaluation metric used is mean column-wise ROC AUC. In other words, the score is the average of the individual AUCs of each predicted column. This means, that false prediction of 'multiple_diseases' will have a huge and disproportionate effect on final evaluation.

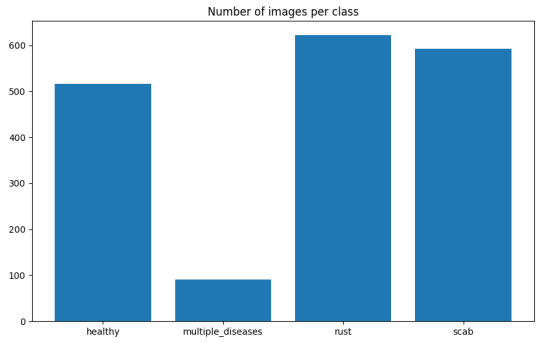

Figure 1: Dataset's class distribution.

2.2 Data Preprocessing

One of big hurdles of this task was the size and diversity of dataset. Because of this it was of great importance to artificially diversify the data in order to not overfit the models on our training images and rather generalize them. Such approaches helped us towards model's robustness.

We tried and tested multiple different augmentation techniques. For the start point we first tested similar augmentation operations, that were used to pretrain each model. With time, using 5-fold cross validation process we compared their effectivness and further fine-tuned the values. We used:

- random resize with interpolation
- center crop
- horizontal flip
- rotation
- color jitter
- normalization

2.3 Models and Architectures

We decided to use 4 different pretrained models. We chose: ResNet50 [2], EfficientNetB4 [3] and 2 different vision transformers, one that takes smaller image dimensions: vit_base_patch16_224 [4] and one with larger one:

vit_base_patch14_reg4_dinov2 [5], to see how size of input images affects the performance.

Our idea was to analyze how each of this models works and compare its results to the others.

For all 4 models we took already pretrained weights, because our small dataset does not allow us to train from scratch. We only want to finetune the models for our specific task. We did that by replacing models' final layers with a linear layer, that has 4 output classes for each of our 4 possible predictions.

The models we chose differ in architecture, depth, number of parameters, input image size, etc.

A very important characteristic, when it came to training time were of course number of parameters and input size of the images.

EfficientNet model is the most lightweight, it has only 19.3M parameters and its specified input dimension is 320x320.

ResNet50 model is a bit larger with 25.6M parameters and input images are of the size 224x224.

Both vision transformers have noticingly more parameters - both have 86.6M. The only difference is between image sizes, where 1 intakes images of 224x224 and the other 518x518.

We also wanted to combine the 4 models and utilize the power of ensemble models. We created a few different solutions.

The most simple idea of those was just averaging the predicted scores of all 4 models and using this as a prediction.

A bit more complex solution was creating a separate neural network that intakes all 4 models' probability predictions and outputs the final combined prediction. This network is being trained from scratch, while the other 4 network weights are frozen and are not changing. Architecture is shown on Figure 2.

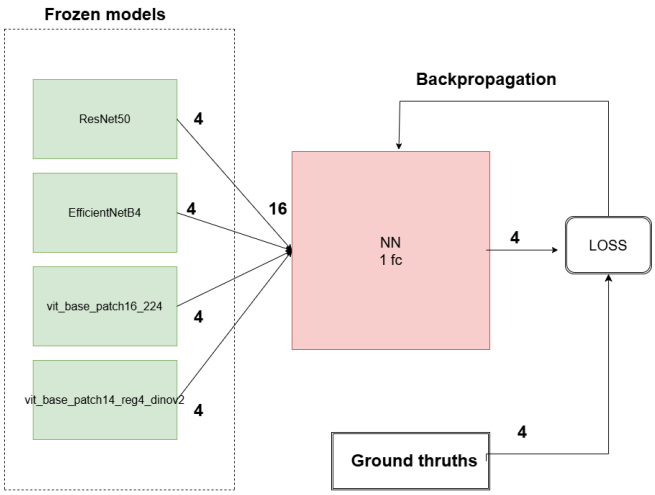

Figure 2: Simple ensemble architecture, probabilities from 4 separately trained models are taken as an input into a simple NN. Net gets trained using ground truths against net's predictions via backpropagation. Dimensions of passed data are labeled with numbers on the image.

Our final idea was to utilize the embedding representations that the models can make. Here we loaded all 4 finetuned models and removed their final (classification) layers. Then when we ran predictions again, the models output the embeddings they create at a layer before classification. Our reasoning for doing this was, that the final embedding is most likely to be the most informative and representative, since the classifications are based on it. Then we concatenated all 4 embedding representations and input them into our simple custom neural network with 3 fully connected layers, 3 ReLU layers, dropout layers and a final fully connected layer for predictions. To train this net from scratch we had to get all the embeddings from 4 models to use them as features in our final network. During the training the 4 models stayed as they were – already finetuned, while the network kept getting updated with backpropagation. The full architecture is shown on Figure 3.

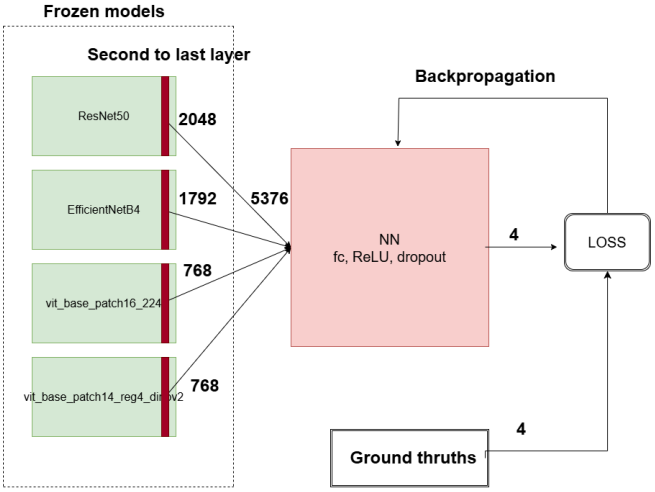

Figure 3: Ensemble architecture, From each model embeddings are extracted by removing the last classification layer. The vectors get concatinated and passed into a few layer NN. Net gets trained using ground truths against net's predictions via backpropagation. Dimensions of passed data are labeled with numbers on the image.

Using all this unique and different models to get the final prediction, we hope that each model encapsulates certain perspective of the data and makes it possible for the final combined model to make a more informed prediction.

Our expectations were, that vision transformers will outperform the more traditional convolutional neural networks and that using larger image sizes would be the most beneficial. Regarding the ensemble methods we expect them to on average outperform the individual models.

2.4 Training Procedure

For our training we used 5-fold cross validation. We used this approach in order to always have a seperate validation set, so we can clearly see when the models start overfitting. Using cross validation we also found some optimal hyperparameters.

We noticed that the optimal starting learning rate was

around 1e-4, except for the vision transformer with larger input size, a slightly higher learning rate worked better.

For all the models we used Adam optimizer with ReduceLROnPlateau scheduler, that lowers the learning rate, when the validation loss starts to stagnate or even worsen. The use of weight decay as a regularization step proved to worsen the results. Unfortunately we had to use a quite low batch size of only 8, in order to not run out of memory, even using the largest GPU's available to us (Arnes Cluster HPC) .

We trained the models for 40 epochs and only saved the model whenever the loss on validation set got lower, in order to not overfit the models. For calculating the loss we tested BCEWithLogitsLoss and CrossEntropyLoss. The latter turned out to be a better loss metric.

On Figures 4, 5, and 6 we show training loss and validation loss during the training process for the models. We can clearly see on each plot, when the models starts to overfit on the training data – we do not save and use those models.

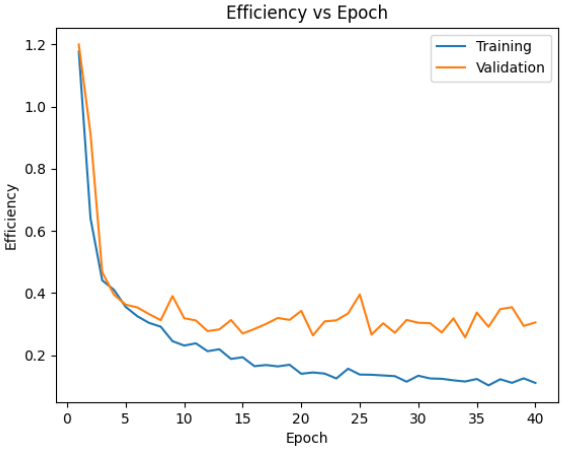

Figure 4: Training and validation loss on EfficentNet

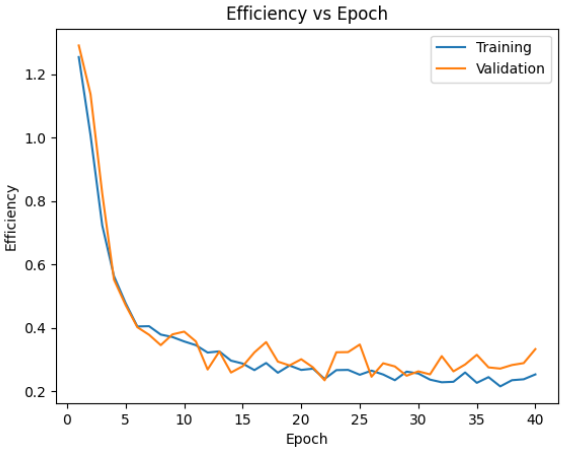

Figure 5: Training and validation loss on ResNet.

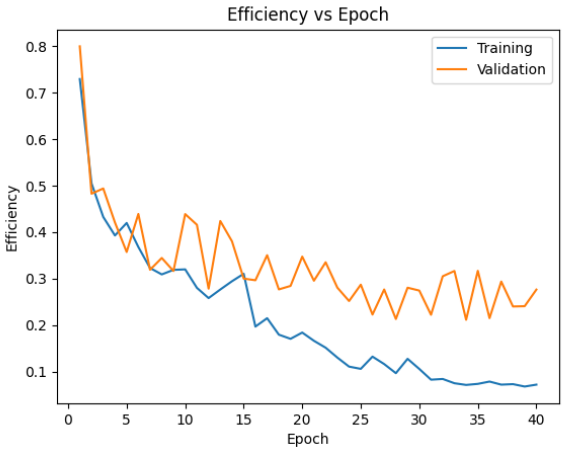

Figure 6: Training and validation loss on VIT with small input images.

Finally at the end we also used an approach with pseudo labels. This is a semi supervised approach, where we can also use unlabeled data. This approach is particularly useful in our case, where our training dataset is pretty small. So once we trained initial models we predicted the class probabilities of each of 4 classes. If a prediction class was chosen with a probability of above a very high number (above 0.95) we added this sample into a training set. We used a high threshold, so only the test samples we were most sure of were added to the training set. We than used this additional data to once again train the models. This approach can be a bit risky since a wrongly classified sample being used to train our models can be detrimental to the classification accuracy.

2.5 Results and analysis

In order to evaluate all our approaches we decided to compare mean column-wise ROC AUC metric. We calculated the results on the training set and test set - test set is further split on public and private set. We report the achieved results of each unique architecture in Table 1

In the next subsections we further comment and try to reason over results.

Individual Models

Between individual models the best performing one is vision transformer with smaller image inputs, with the score of 0.940 on the private test set. We expected transformers to outperform the CNNs, so this was was not a particular surprise. We were still impressed, that the one with the smallest input image produces such good results.

The biggest surprise to us was the performance of the other visual transformer, with the largest input images. We believed, that it would clearly outperform the other 3 models, but as we can see the opposite happened. We are not entirely sure why that happened, but our guess would be suboptimal training parameters – we noticed it trains slower and increased the learning rate, but maybe not enough. Or perhaps it was pretrained on a very different images and had a

Table 1: Performance comparison of different models and ensembles. All the metrics are mean column-wise ROC AUC. VIT is vision transformer, avg. score is avereging every models prediction, pred. + NN is the approach of 1 layer NN making predictions based on every models prediction, embedd. + NN is training the NN model on 4 models embeddings and + pseudo means using the pseudo labels. for training

Approach	Train	Public Test	Private Test
EfficientNet	0.969	0.937	0.939
ResNet	0.962	0.930	0.924
VIT small	0.975	0.939	0.940
VIT large	0.908	0.868	0.857
avg. score	0.988	0.951	0.953
pred. + NN	0.990	0.962	0.959
embedd. + NN	0.986	0.955	0.957
pred. + NN + pseudo	0.990	0.963	0.955
embedd. + NN + pseudo	0.986	0.957	0.957

hard time finetuning to our set. Another factor might be low image count, since transformers might need a bit more data to further improve.

The CNN models performed as expected, their scores were fairly similar and we were quite satisfied with their performance, based on the small number of parameters and quick training time – especially the lightweight efficientNet.

One of our hypothesis of larger images sizes clearly improving the resutls proved out to be false.

Ensemble Architectures

Interesting results were obtained by only just simply averaging the results. This technique saw quite a significant increase in performance on all the given datasets.

The other method of training a separate simple NN with just a fully connected layer for final predictions. Training this network we had to keep in mind not to overfit it – we again used validation set and low number of epochs. This outperforms the simple average method by a bit. This makes sense, since putting 1 layer NN is very similar to just employing a smarter or weighted averaging, where the net learns, how much each model contributes to the correct prediction and weights it accordingly.

Our final method of taking embeddings from each model and training a 10 layer NN on top of that turned out to not be as effective as we thought. It slightly falls behind the simpler method of 1 layer prediction NN on all 4 model predictions. Even though the difference is not drastic we still expected better results and thought this will be our most successful method. The problem could still lie in us not utilizing the embeddings to their fullest potential and making a suboptimal NN architecture that intakes the embeddings. Different shapes of embeddings could also have its effect – larger embeddigns might contribute too much over smaller ones. We expect, that if we really fine-tuned this model and its architecture it would outperform the other ones since it intakes the most amount of data on which it can make the prediction. Another improvement could be also taking embedding from more layers than just one before last, but rather concatinating more of them.

On the final 2 mentioned methods we also used the before mentioned and explained pseudo labels. This gave us a small amount of new training examples. The results using this method on average get slightly improved, but not by much, due to the small increase of training samples. We could have lowered the confidence level, but that would mean that predictions we are not sure of would have a say in our models training, which is not ideal. The method still seems a beneficial thing, when the training data is scarce. In our case due to small number of such cases the effect of this method was minimal, but in certain situations it could improve the models by a lot.

Based on the provided results we proved that ensemble methods can utilize different aspects, that every model focuses on and is almost always beneficial to the performance. Such methods are especially effectitive considering the training time of this final NNs (or even just prediction averaging). Also because of their smaller size they can get trained very quickly

3 Conclusion and future work

In our report we showed and presented multiple of our solutions to solving the problem of plant pathology. We used different models and ensemble combinations and compared them between each other. We clearly described our approaches and methodologies. We also tried to explain and provide additional reasoning to our achieved results. We proved the power of ensemble methods and their effectiveness. The project was really nice to work on and we learned a lot.

However there still are some possible improvements that could be done to further improve our approaches and make the models more generalized and robust. A big part would be further parameter hyper-tuning, where many values should be tried for various parameters. But the biggest improvement, that would address our problem of small dataset would be creating a GAN (generative adverserial network), that would be trained on the given training data and would generate some additional, but different data to use for training. Of course such network should be evaluated and proven to generate high quality results, that reflect the given label, otherwise we would only get more bad and noisy data, which could possibly be worse for our models.

Another possible improvement is handling the class imbalance of 'multiple diseases', which is very underrepresented. We could weight such samples more and therefore inflate importance of each individual sample from this class.

References

[1] M. S. D. Christine Kaeser-Chen, Fruit Pathology, "Plant pathology 2020 - fgvc7," 2020. [Online]. Available: https://kaggle.com/competitions/plant-pathology-2020-fgvc7

[2] R. Wightman, H. Touvron, and H. Jégou, "Resnet strikes back: An improved training procedure in timm," 2021.

[3] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *ICLR*, 2021.

[5] M. Oquab, T. Darcet, T. Moutakanni, H. V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, R. Howes, P.-Y. Huang, H. Xu, V. Sharma, S.-W. Li, W. Galuba, M. Rabbat, M. Assran, N. Ballas, G. Synnaeve, I. Misra, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "Dinov2: Learning robust visual features without supervision," 2023.