POLITECNICO
DI MILANO

AN2DL Challenge 1

Authors:
Sanjay Shivakumar Manohar, 10864950
Satvik Bisht, 10886954
Duggaraju Venkata Sraavya, 10879043

Plant leaves classification with CNNs

1 Introduction

In this task, we are required to classify plants using neural network architectures that are divided into two categories according to their state of health. It is a binary classification problem, so the goal is to predict the correct class label in {0, 1}.

2 Model from Scratch

In the first phase of the challenge, we decided to experiment with the design of a simple Convolutional Neural Network with a structure inspired by common examples in the literature and seen in the lectures (Lenet, VGG9). In this phase, we used the data directly without any preprocessing with 3 convolutional layers with the respective filters (32,64,128) having Relu as an activation function, each one followed by a max-pooling layer. A Global Average Pooling layer was introduced after the convolution part as input to a Fully connected neural network composed of the output layer with 2 neurons, which had Softmax activation function for classification. We proceeded in training the model using (80 ÷ 20) split ratio for the training and validation set, accuracy as a metric, and Categorical cross-entropy as a loss function. With this configuration our first model scored, as expected, 51% in the Codalab leaderboard. We were able to identify that the model was overfitting as we were getting 80% on our local machine.

After this, we tried different architectures like having 6 convolutional layers with the respective filters (32,32,64,64,128,128). After every two layers, we had a max pooling layer followed by a similar approach in the previous model. We split the dataset into training sets (80%) and validation sets (20%) with stratification so that the proportions between classes are the same in the training and validation set and started using Early stopping callback to save our best scoring model. Also, a rescaling of 255 in the input images was introduced to regularize the data. After these modifications, we ended up with the definitive version of our model built from scratch, which improved the overall evaluation metrics, leading to a validation accuracy of around 60% in Codalab while the accuracy in our local environment was 82%. We again faced the same problem of overfitting

3. Data preprocessing and data augmentation

We found out that the dataset had some outliers as cartoon images and it was unbalanced with a 62:38 healthy: unhealthy ratio. Our first challenge was to remove the cartoon images. We wrote a Python code to display the images randomly and noted the index of cartoon images. Then through a loop, all matching numpy values to that of the cartoon image were removed from the dataset.

Initially, data augmentation was done to balance the classes. We used spatial transformations (vertical, horizontal RandomFlips, and Random Translations) on every 3rd and 4th image unhealthy image (it was a random factor to reduce the imbalance by working on close to 850 unhealthy images) using sequential layers in Keras. We also up-sampled the training dataset using the same spatial transformations as mentioned above, which increased the training data from 4927 to 7927 images.

4. Transfer Learning and fine-tuning

Now, we try different well-known pre-trained networks available in Keras Applications - MobileNet, VGG16, AlexNet, Resnet, Xception, and ConvNext and compare the accuracy obtained to fine-tune only the most performing ones. First, we removed the top layer to give our customized input and make the convolutional layer of the model not trainable, did global pooling, and added the output layer with 2 neurons. By doing this, initially, we had just a few parameters from the last layers to be trained. By performing the above transfer learning all the models had an accuracy of around 80% on the test data. Then we added custom dense layers to the model and tried transfer learning again with no fine tuning on the base model. Performance didn't seem to improve much and the model started to overfit.

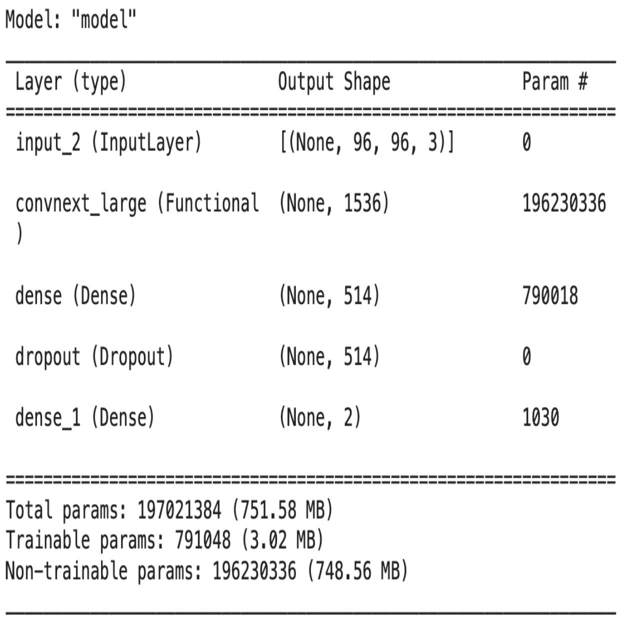
We then tried fine tuning on the models, by freezing the initial convolution layers and unfreezing just the last few. As the deeper networks contain higher-level features in the last layers, it was decided to fine-tune from here and leave the lower-level features untouched. This improved the accuracy compared to just transfer learning but, in some cases, the model was overfitting owing to fewer image data and larger parameters. To mention a particular instance, our fine-tuned Resnet50 model had an accuracy of 93% on test data, 97% on the training data but it just managed an accuracy of 79% with the test set in CodaLab in the development phase. We tried smote upsampling and a 3-fold validation technique to improve the Resnet50 model but didn't see any improvement in accuracy with the test data.

We also tried a technique similar to model ensemble where we made an AND logic between results from Mobilenet fine-tuned and Resnet fine-tuned models. We did AND logic as the unhealthy leaf images in the training set were less, so to output a prediction as unhealthy when only both models predicted them as unhealthy. This technique gave an accuracy of 81% and **a recall of 100%** in the codalab test set during the development phase indicating we had no false negatives.

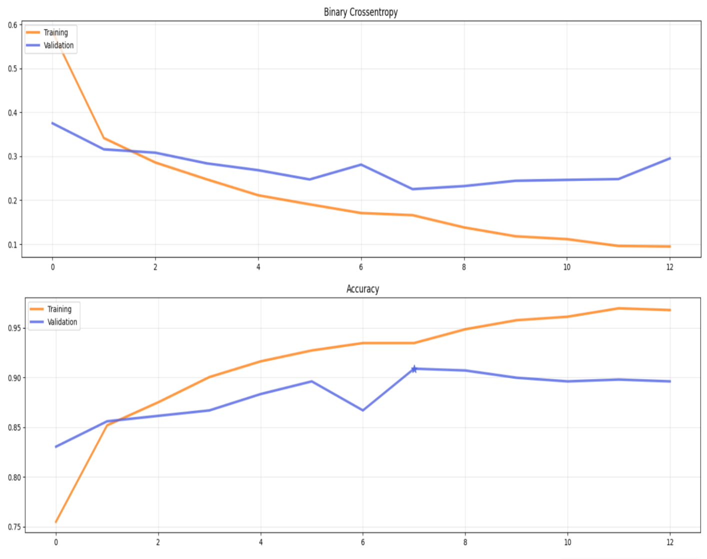
Finally, we wanted to experiment with ConvNext models. We started with the ConvNext Base model, and followed the same preprocessing and data augmentation techniques. All the other models required some kind of specific preprocessing for input data like Resnet preprocess or Mobile net preprocess to convert from RGB to BGR, however, with ConvNext none of them were required. Like before, we fine-tuned the ConvNext Base model with the last few layers and it seemed to converge quickly in just 7 or 8 epochs. The early stopping and reduction of LR plateau scheduler callbacks made sure the model wasn't overfitting. This model locally gave us an accuracy of 95% and when submitted during the development phase, it had an accuracy of 91% which is our highest score for the development phase in terms of accuracy.

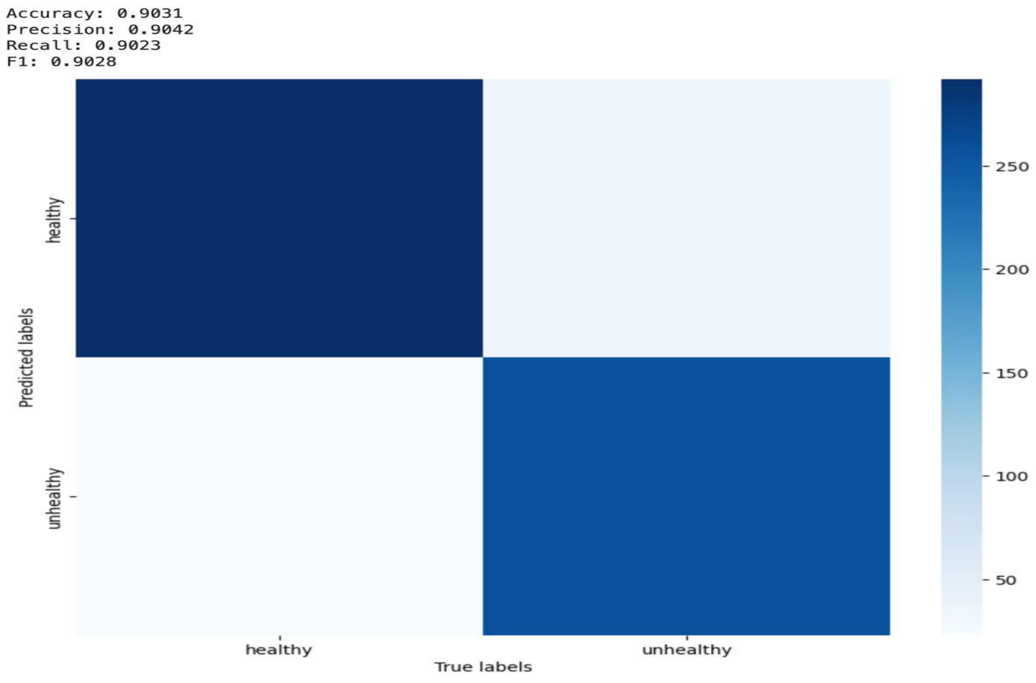
We used our best model from the development phase and it received an accuracy of 80% in the CodaLab platform. We then tried to implement ConvNextLarge using the same preprocessing and data augmentation. We made the convolutional layers not trainable using Global average pooling to flatten the convolutional layer and then added 1 dense layer with 512 neurons, softmax activation following with a dropout of 0.3. We used binary-cross entropy and adam as loss functions and optimizers respectively trained the model for 40 epochs and a batch size of 160 which helped us to get an accuracy of **82% in the CodaLab platform**. We also tried fine-tuning the model using different learning rates

(1e-5) and tried freezing different numbers of convolution layers (150, 200, 230). The model was getting 90% in our local machine but we were not able to test it on the CodaLab. To try and further improve the accuracy we also tried the ConvNextXLarge model and got 90% accuracy during development but didn't get final results on CodaLab before the deadline. Hence our best accuracy was **82% in the final phase** and **91% in the development phase** in CodaLab.



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 96, 96, 3)]       0

 convnext_large (Functional  (None, 1536)              196230336
 )

 dense (Dense)               (None, 514)               790018

 dropout (Dropout)           (None, 514)               0

 dense_1 (Dense)             (None, 2)                 1030

=================================================================
Total params: 197021384 (751.58 MB)
Trainable params: 791048 (3.02 MB)
Non-trainable params: 196230336 (748.56 MB)
_________________________________________________________________
```





```
Accuracy: 0.9031
Precision: 0.9042
Recall: 0.9023
F1: 0.9028
```

References:

Keras Website : https://keras.io/api/applications/
Keras docs: https://keras.io/
Medium Links:
1.https://medium.com/swlh/how-to-use-smote-for-dealing-with-imbalanced-image-dataset-for-solving-classification-problems-3aba7d2b9cad
2.https://medium.com/augmented-startups/convnext-the-return-of-convolution-networks-e70cbe8dabcc

Contributions:

Satvik Bisht (10886954):

1. Data Augmentation with different spatial transformations rotation, translation, zoom, etc.
2. Build models from scratch Lenet, VGG9.
3. Transfer Learning using Xception.
4. Transfer learning using ConvnextTLarge used early stopping and fine-tuning the model with different learning rates, and the number of freezing layers. Experimented by adding different numbers of dense layers and drop-outs.
5. Performed some data augmentation with an image generator as well.

Sanjay(10864950):

1. Data Preprocessing - Removing cartoon images.
2. Data Augmentation with Sequential layers and spatial transforms to balance dataset and increase dataset.
3. Transfer learning with Mobile net. Transfer learning and Fine tuning Resnet50, Xception, and ConvNext base, using early stopping and reduction on LR plateau callbacks.
4. Trying a model ensemble-like technique, producing a good recall.

Sraavya (10879043):

1. Data Augmentation with smote upsampling and different spatial transformations and k-fold validation on ResNet model.
2. Build AlexNet Model from scratch.
3. Tried ensemble learning with multiple models.
4. Transfer Learning using ConvNextXLarge used early stopping and fine-tuning.