Plant Pathology - Identify the category of foliar diseases in apple trees

1st Ashish Bodhankar
MS in Data Science
Stevens Institute of Technology
Hoboken, New Jersey, USA
abodhak@stevens.edu

2nd Joshnanjani Mataparthi
MS in Data Science
Stevens Institute of Technology
Hoboken, New Jersey, USA
mjoshnan@stevens.edu

3rd Vineeta Dmello
MS in Data Science
Stevens Institute of Technology
Hoboken, New Jersey, USA
vdmello@stevens.edu

***Abstract*—Apple orchards in the United States are under constant threat from many pathogens and insects. Appropriate and timely deployment of disease management depends on early disease detection. Incorrect and delayed diagnosis can result in either excessive or inadequate use of chemicals, with increased production costs and increased environmental and health impacts.**

We used the expert-annotated data set of 18,600 apple leaf images provided to the Kaggle community for the Plant Pathology Challenge as part of the Fine-Grained Visual Categorization (FGVC) workshop. Foliar disease labels included apple scab, cedar apple rust, complex diseases, frog eye leaf spot, powdery mildew, and healthy leaves. Participants were asked to use the image data set to train a machine learning model to classify disease categories and develop an algorithm for disease severity quantification. We trained a baseline sequential CNN model and achieved a validation set accuracy and F-1 Score of 83% and 77% respectively. This was our best model as opposed to two other classic CNN architectures (Transfer Learning) fine-tuned for this training set. InceptionResNetV2 and EfficientNetB1 achieved a validation F1-score of 68.74% and 73.02% respectively. This was the opposite of what we expected. We believed the transfer learning models would outperform our baseline model which is a much simpler and smaller network.

This data set will contribute toward development and deployment of machine learning–based automated plant disease classification algorithms to ultimately realize fast and accurate disease detection. The organization will continue to add images to the pilot data set for a larger, more comprehensive expert-annotated data set for future Kaggle competitions and to explore more advanced methods for disease classification and quantification.

I. Introduction

For the 2021 spring semester, our team has decided to choose an active Kaggle competition dataset to work on the group project. "Plant Pathology 2021 – FGVC8" is a research Code Competition released on March 15th, 2021, currently ongoing, and, will continue till 26th May before it eventually ends. This competition is part of the Fine-Grained Visual Categorization FGVC8 workshop at the Computer Vision and Pattern Recognition Conference CVPR 2021.

Apples are one of the most important temperate fruit crops in the world. Foliar (leaf) diseases pose a major threat to the overall productivity and quality of apple orchards. The current process for disease diagnosis in apple orchards is based on manual scouting by humans, which is time-consuming and expensive.

Although computer vision-based models have shown promise for plant disease identification, there are some limitations that need to be addressed. Large variations in visual symptoms of a single disease across different apple cultivars, or new varieties that originated under cultivation, are major challenges for computer vision-based disease identification. These variations arise from differences in natural and image capturing environments, for example, leaf color and leaf morphology, the age of infected tissues, non-uniform image background, and different light illumination during imaging etc.

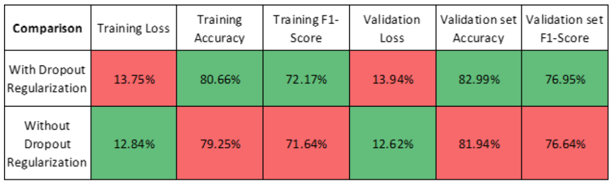
Last year's Plant Pathology 2020-FGVC7 challenge competition had a pilot dataset of 3,651 RGB images of foliar disease of apples. For Plant Pathology 2021-FGVC8, they have significantly increased the number of foliar disease images and added additional disease categories. This year's dataset contains approximately 23,000 high-quality RGB images of apple foliar diseases, including a large expert-annotated disease dataset. 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.

Specific Objective: The main objective of this competition is to develop Machine Learning-based models to accurately classify a given leaf image from the test dataset to a particular disease category, and to identify an individual disease from multiple disease symptoms on a single leaf.

Dataset Description: The compressed parent file contains 18,600 training images (.jpeg) along with a CSV file that marks a label of the target classes to every image in the training set. The target classes are a space delimited list of all diseases found in each training set image. Unhealthy leaves with too many diseases to classify visually will have the "complex" class and may also have a subset of the diseases identified. For now, the 5000 test images are hidden and will be available once we submit our notebook on Kaggle. For now, we have decided to split the available 18,600 images further into Train and Validation set for building and evaluating Convolution Neural Networks.

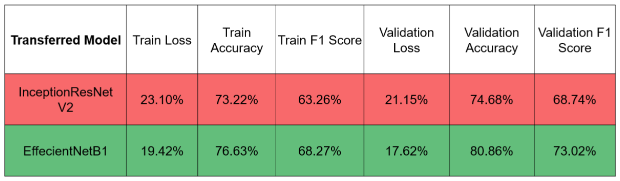
Machine learning algorithms we used to solve the prob-

lem: We built a sequential CNN from scratch and compared its performance to EfficientNetB7 and InceptionResnetV2, finetuned to our dataset. Adam optimization algorithm is considered to train these models on mini batches along with "Binary Cross Entropy" for the loss. Firstly trained a baseline sequential CNN and observed the following accuracy and F1 scores for our model. We trained the same network twice, once without any dropout and the second time with a dropout layer to quantify the effect of regularization on the model performance. Below is the summary of our model's evaluation metrics.



Comparison	Training Loss	Training Accuracy	Training F1-Score	Validation Loss	Validation set Accuracy	Validation set F1-Score
With Dropout Regularization	13.75%	80.66%	72.17%	13.94%	82.99%	76.95%
Without Dropout Regularization	12.84%	79.25%	71.64%	12.62%	81.94%	76.64%

Next we trained EfficientNetB7 and InceptionResnetV2 using transfer learning and expected them to out perform our baseline model but found the opposite results. The performance metrics of these networks on the same pre-processed data gave the following results:-



Transferred Model	Train Loss	Train Accuracy	Train F1 Score	Validation Loss	Validation Accuracy	Validation F1 Score
InceptionResNet V2	23.10%	73.22%	63.26%	21.15%	74.68%	68.74%
EffecientNetB1	19.42%	76.63%	68.27%	17.62%	80.86%	73.02%

Since this is still an active competition as of now, Leaderboard scores are not published for this year's plant pathology competition. But as per the details and background information on the last year's Kaggle competition, a performance of 97 percentage accuracy on held-out test set was reported from off-the-shelf transferred models. We could not achieve such high accuracy due to reasons mentioned in the next section under "II. Related Work".

II. Related Work

We could not achieve the high accuracy reported by last year's competition winners due to some of the following reasons: -

1.) Our preprocessing was limited to identifying the exact duplicates and merely resizing the images. Whereas the 1st place winners of the 2020-FGVC7 Kaggle competition used a much more advance augmentation technique and trained the model on higher resolution images as compared to us.

2.) Not the right choice of last few layers that we added to the Transferred architectures.

3.) Moreover, we have more classes this time as compared to previous year's disease labels. And unlike before, one image can be infected with more than one disease. This increased the complexity further as we cannot simply use a softmax (Multi Class classification) function. This is a multi label classification problem.

4.) They also used the "knowledge distillation" method to solve the problem of noisy labels in the training data by first training a 5-fold model and get out-of-fold results about valid dataset, and then mix the out-of-fold results and ground truth by 3:7, as the labels of a new training model.

5.) Due to limited computation and GPU capacity we had to resize the (4000px,3000px) images to (256px,256px). This has surely impacted our model's accuracy by reducing the bias because the training features considerable reduces.

6.) And we only used Keras' inbuilt data augmentation functionality and no custom fancy augmentation techniques.

7.) Study related literature to acquire domain knowledge - This area we clearly lack the necessary expertise.

8.) This data is clearly imbalanced. A need for stratified sampling of the training and validation set can help improve the performance considerably. We did not include stratified folds while creating mini batches during training.

Additional details and further background information on the 'Plant Pathology 2020 challenge' can be found here:-

Thapa, Ranjita; Zhang, Kai; Snavely, Noah; Belongie, Serge; Khan, Awais. The Plant Pathology Challenge 2020 data set to classify foliar disease of apples. Applications in Plant Sciences, 8 (9), 2020.

Additional references around Deep learning architectures and Neural networks that we used in this project are as follow: -

[1] - Google AI Blog

Going deeper with convolutions - [1]

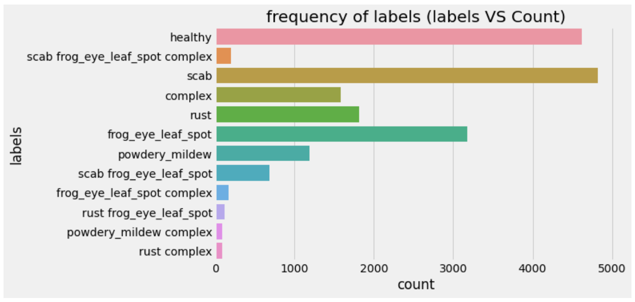
Deep residual learning for image recognition - [2]

III. Our Solution

This section elaborates our solution to the problem.

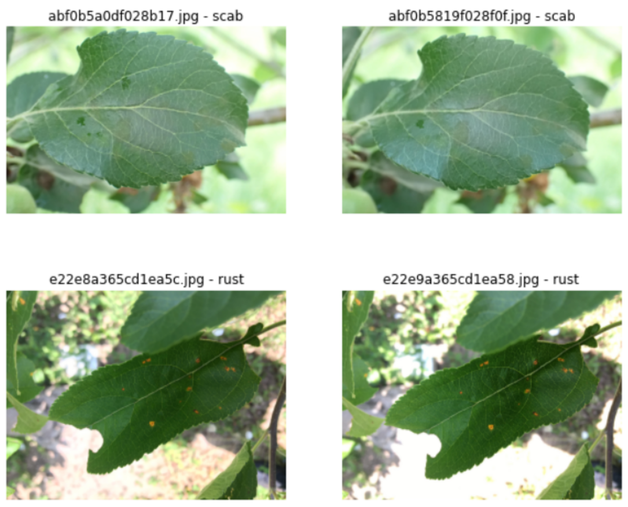
A. Description of Dataset

We split the provided 18,600 training images in 80%-20% ratio for the training and validation purpose. The histogram shown below gives us some insights on the how the images are labelled across the entire set of 18,600 images. Some of these labels are just a combination of multiple diseases classified to one leaf. Hence, there are not 12 labels but it's just 6 labels (5 diseases). Now, since we know that one leaf can have multiple diseases, this clearly is the case of a multi label classification and not a multi class classification problem (Plant Pathology 2020 - FGVC7).



Also, since each image is quite big in the dataset and since during training, the model predicts with GPU and loads and augments the image with CPU. We cannot completely activate the GPU if the CPU always takes time to load the high-resolution images, resulting in breaking the continuity, and affecting proper GPU utilization. And since we are loading and training our model at the same time it takes some more extra time to load these images. And hence, a need to resize these images was an important part of the preprocessing to eventually train the model faster. We saved the downscaled images in a different folder and use these images thereafter.

The next task was to identify the noise in the resized images. Duplicates are always harmful for training process - differently labeled duplicates produce noise in the dataset, while equally labeled duplicates lead to data leakage. For this project we searched through the plant pathology 2021 dataset with the ”Image Hash” library, revealing around 83 duplicates. Two examples are shown below.

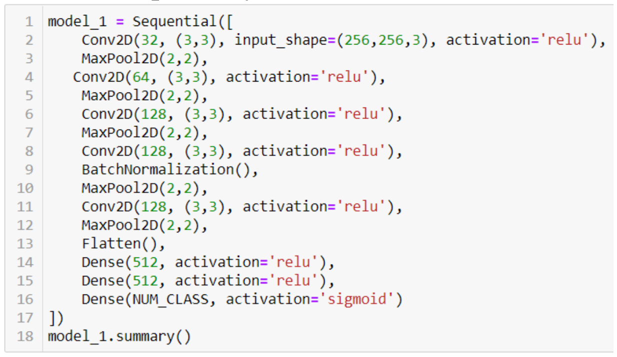

abf0b5a0df028b17.jpg - scab
abf0b5819f028f0f.jpg - scab
e22e8a365cd1ea5c.jpg - rust
e22e9a365cd1ea58.jpg - rust

We then encoded our categorical labels to numbers before modeling the data. The keras’ ImageDataGenerator class provides the image augmentation functionality which we used to expand the training dataset (not the validation set) to improve the performance and ability of the model to generalize. Transformations including a range of operations like re-scaling, rotation, width and height shift, zoom and flip were considered in our case. The main benefit of using the ImageDataGenerator class is that it is designed to provide real-time data augmentation. Meaning it is generating augmented images on the fly while our model is still in the training stage. It also ensures that the model receives new variations of the images at each epoch. But it only returns the transformed images and does not add it to the original corpus of images. Another advantage is that it requires lower memory usage. This is so because without using this class, we load all the images at once. But on using it, we are loading the images in batches which saves a lot of memory.

B. Machine Learning Algorithms

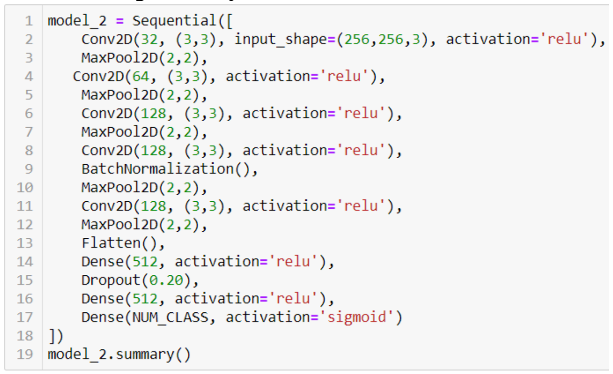
We decided to first build a sequential CNN from scratch as our baseline model that includes a Batch Normalization layer to help speed up the training. To specifically understand the effect of Regularization on the model’s performance, we trained the model, first, without a dropout layer. And the second time, again, with a Dropout layer. In both the cases we tried to maintain all other factors same to make a sensible comparison.

Without Dropout Layer



```
model_1 = Sequential([
    Conv2D(32, (3,3), input_shape=(256,256,3), activation='relu'),
    MaxPool2D(2,2),
   Conv2D(64, (3,3), activation='relu'),
    MaxPool2D(2,2),
    Conv2D(128, (3,3), activation='relu'),
    MaxPool2D(2,2),
    Conv2D(128, (3,3), activation='relu'),
    BatchNormalization(),
    MaxPool2D(2,2),
    Conv2D(128, (3,3), activation='relu'),
    MaxPool2D(2,2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(512, activation='relu'),
    Dense(NUM_CLASS, activation='sigmoid')
])
model_1.summary()
```

With a Dropout Layer



```
model_2 = Sequential([
    Conv2D(32, (3,3), input_shape=(256,256,3), activation='relu'),
    MaxPool2D(2,2),
   Conv2D(64, (3,3), activation='relu'),
    MaxPool2D(2,2),
    Conv2D(128, (3,3), activation='relu'),
    MaxPool2D(2,2),
    Conv2D(128, (3,3), activation='relu'),
    BatchNormalization(),
    MaxPool2D(2,2),
    Conv2D(128, (3,3), activation='relu'),
    MaxPool2D(2,2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.20),
    Dense(512, activation='relu'),
    Dense(NUM_CLASS, activation='sigmoid')
])
model_2.summary()
```

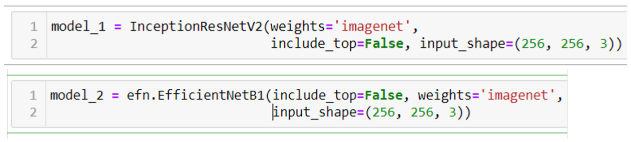
Both the models were compiled using the “Adam” optimizer and “Binary Cross Entropy” for the loss. This is clearly a case of “multi label” classification problem and not a “multi class” classification because each leaf image can be infected with more than one foliar disease. Hence, we did not use a “Softmax” function for this reason.

“Accuracy” and “F1-score” were the two “optimization metrics” we considered primarily. And the “speed of the model’s prediction on one example” was considered as the secondary “satisficing metric”. Next, we decide to train two “off the shelf classic CNN architectures” using “Transfer Learning” to make a comparison of our baseline model and gauge the differences. Here we used weights from the InceptionResnetV2 and EfficientNetB01 networks pretrained on ”ImageNet” dataset and fine-tuned these weights on our training set. The are two reasons why we chose InceptionResNetV2, firstly, the presence of ”Residual” blocks. Residual blocks allows the flow of memory (or information) from initial layers to layers deeper than the immediate next layer. Identity function is easy for residual blocks to learn. This helps in training very deep convolution neural networks, according to the original paper on ResNets [2]. It is a common belief that the accuracy of NN increases with an increase in the number of Layers. But there is a limit to the number of layers added that result in accuracy improvement. As the network grows larger, the curse of dimensionality problem becomes prominent. Also, the vanishing and exploding gradient problem has its effect. These cause difficulties for the network to learn basic functions like the "identity" functions. Skip connections or residual connections help us learn these identity functions even though we increase the number of layers.

And the second reason for choosing InceptionResNetV2 is because this network also has ”Inception Bottleneck Layers” which helps in reducing the number of computation steps. In a CNN (such as Google's Inception network), bottleneck layers are added to reduce the number of feature maps (aka ”channels”) in the network, which otherwise tend to increase in each layer. This is achieved by using 1x1 convolutions with less output channels than input channels.

Similarly, our reason, for choosing EfficientNetB01 was that we were inspired by an article release by GoogleAI (link already mentioned earlier in the ”Related Work” section) which claims that EfficientNet models achieve both higher accuracy and better efficiency over existing CNNs (On ImageNet dataset), reducing parameter size and FLOPS by an order of magnitude. It is based on more principled method to scale up a CNN to obtain better accuracy and efficiency. By providing significant improvements to model efficiency, Google expect EfficientNets could potentially serve as a new foundation for future computer vision tasks.

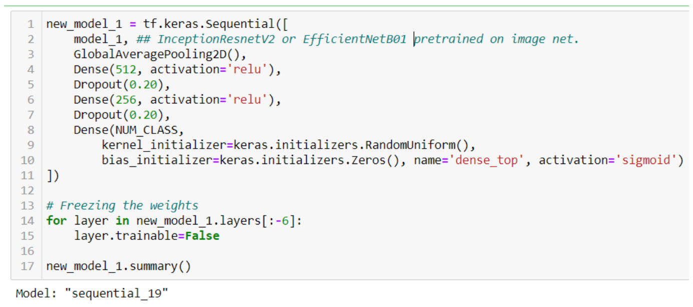
As you can see that, as we import the pre-trained weights of the two architectures, we don't consider the last layer as that layer outputs the classes of the original ImageNet dataset.



```
model_1 = InceptionResNetV2(weights='imagenet',
                            include_top=False, input_shape=(256, 256, 3))
```

```
model_2 = efn.EfficientNetB1(include_top=False, weights='imagenet',
                            input_shape=(256, 256, 3))
```

Once we had our pre-trained weights, we froze them and only trained the weights of a few extra layers that are added towards the end. As shown below, you can see the nature of deep layers we added after the transferred weights. The output layer predicts the same number of classes as required for FGVC-2021 dataset. We added the same last few layers after both the sets of transferred weights.



```
new_model_1 = tf.keras.Sequential([
    model_1, ## InceptionResnetV2 or EfficientNetB01 pretrained on image net.
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.20),
    Dense(256, activation='relu'),
    Dropout(0.20),
    Dense(NUM_CLASS,
        kernel_initializer=keras.initializers.RandomUniform(),
        bias_initializer=keras.initializers.Zeros(), name='dense_top', activation='sigmoid')
])

# Freezing the weights
for layer in new_model_1.layers[:-6]:
    layer.trainable=False

new_model_1.summary()
```

```
Model: "sequential_19"
```

We imported a total of 55,256,293 non trainable parameters from the InceptionResNetV2 and trained only 919,557 extra parameters. And similarly, for EfficientnetB01, a total of 7,363,717 non-trainable parameters and 788,485 trainable parameters.

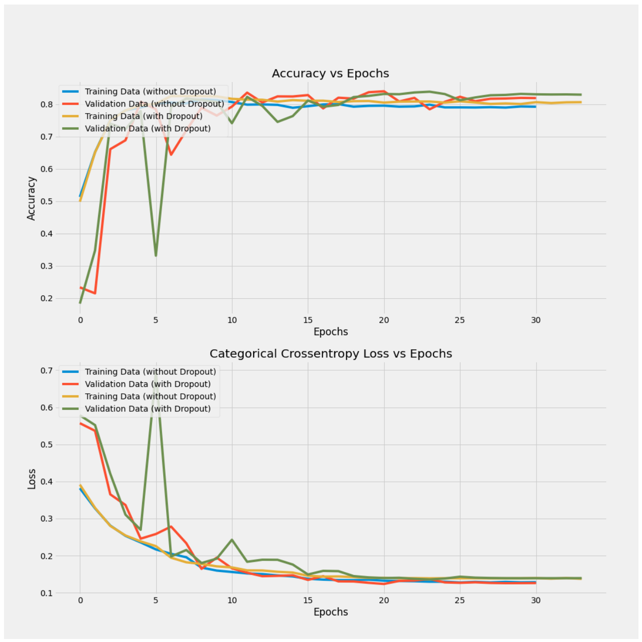
C. Implementation Details

As mentioned above, for all the three models, we used the Adaptive Momentum Estimation (Adam) algorithm as our optimizer and we used mini-batches of size = 128 images per iteration within each epoch. All the times the models were trained for 40 epochs. The ”ReduceLROnPlateau” class provides a functionality to improve the learning rate by reducing the metric by a factor of 0.2 in case it stagnates over three epochs (patience = 3).

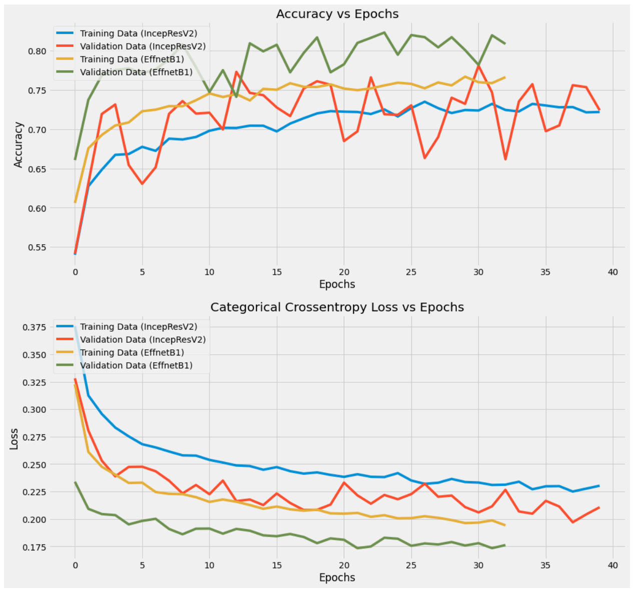
It was observed that the loss was not improving any further and hence we implemented ”Early Stopping” at the 31st epoch for the first network (without dropout) and at the 34th epoch for the second network (with dropout). For the third (IncepResV2) and the fourth (EffNetB01) we continued till the 40^{th} epoch.

And as per [3] and [1], using batch normalization improves accuracy with only a small penalty for training time and enables higher learning rates by reducing internal covariate shift. And hence we made sure both first and the second networks have a batch normalization layer by default. And as per the recommendations by author of the Adam paper we used the hyperparameter value of 0.9 for the momentum term, 0.999 for the RMS prop term and the bias correction term, epsilon, equal to 1e-8. The learning rate alpha, we took as 0.001. Same initial learning rate all 4 times.

The second network performed better than the first network (without a dropout layer). As per the research paper on dropout, published in the Journal of Machine Learning by Geoffrey Hinton and his team [4], recommends a dropout percentage between 20%-50%. We knocked out 20% of the hidden units in the dropout layer while training the model. This clearly shows the positive effect of regularization in reducing the variance of the model on the validation set. The below figure is a visual representation of how the learning rate and F1-score improved with each epoch, for the first two networks.



And similar graph between the two transferred learning models are shown below. Here and in the next section we see that the EfficientNetB01 performed better than Inception-ResNetV2.



IV. Comparison

Our baseline model performed better than the models trained through transfer learning. Previous year, off-the-shelf models achieved close to 97% F1 score. In fact, even in our case we expected our models to outperform the base line model but observed the opposite. Reason for this, we believe could probably be:-

1.) Not the right choice of trainable layers added after the transferred network.

2.) Network too complex and data not sufficient. But this might not be true because 97% F1 score was observed on transferred learning models last year.

3.) Not enough Data Augmentation.

4.) Extreme downscaling of images.

5.) Lack of domain knowledge that was probably used to better preprocess the data.

Below is the final comparison.

Comparison	Training Loss	Training Accuracy	Training F1-Score	Validation Loss	Validation set Accuracy	Validation set F1-Score
With Dropout Regularization	13.75%	80.66%	72.17%	13.94%	82.99%	76.95%
InceptionResNet V2	23.10%	73.22%	63.26%	21.15%	74.68%	68.74%
EffecientNetB1	19.42%	76.63%	68.27%	17.62%	80.86%	73.02%

V. Future Directions

Some of the potential directions for further improving of the performance, we believe, are as follows:-

1.) Data is clearly skewed. Some classes have way more images than other. Stratified Sampling could also improve any model's accuracy by a considerable amount.

2.) Understanding the right way, if any, to scale a CNN architecture to obtain better accuracy and efficiency.

3.) A grid search through various hyperparameters.

VI. Conclusion

We achieved a best accuracy of 84% from our best model. This shows that there is enough scope of improvement in our implementation of Deep learning methods to solve object detection and image recognition problems. We plan to educate ourselves on how to better pre-process the data and also develop a more principled ways of expanding Neural Networks.

The foliar disease symptom images in our image data set represent the complexities that potentially exist in real-life scenarios. The machine learning models developed using such databases with increasing training size every year, will potentially reduce overfitting of models and could be efficient for accurate classification of apple diseases. The organization will continue adding more images captured using a range of angles, lighting, and distances to our pilot data set to build an even larger, more comprehensive expert-annotated data set.

References

[1] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

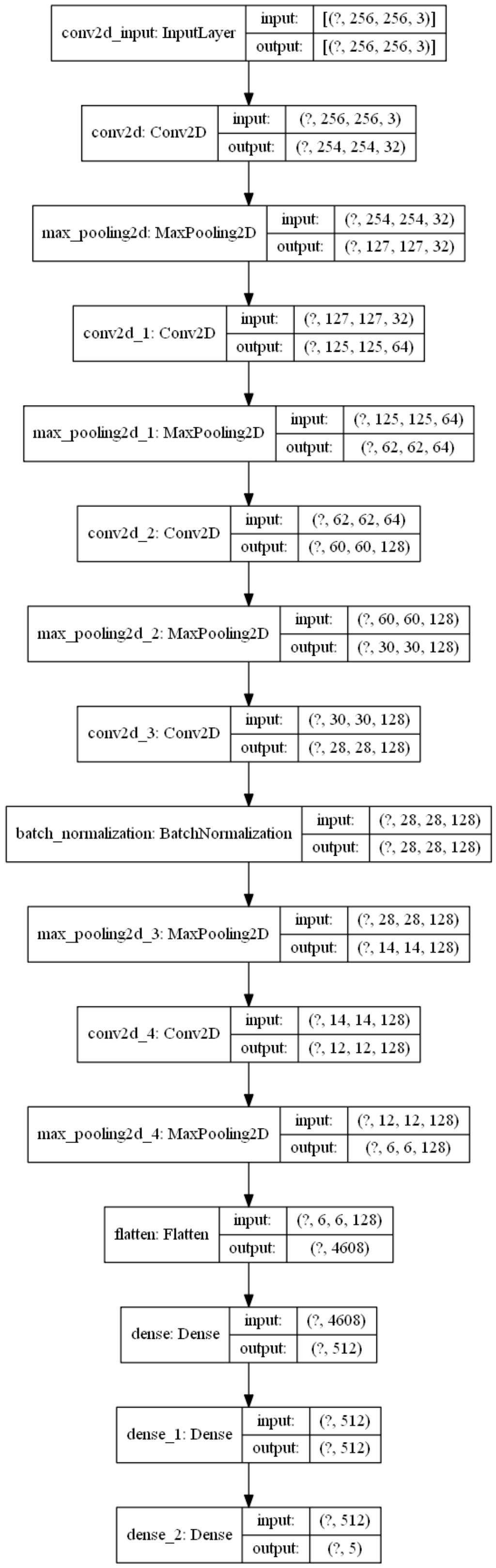
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] C. Garbin, X. Zhu, and O. Marques, “Dropout vs. batch normalization: An empirical study of their impact to deep learning,” *Multimedia Tools and Applications*, pp. 1–39, 2020.

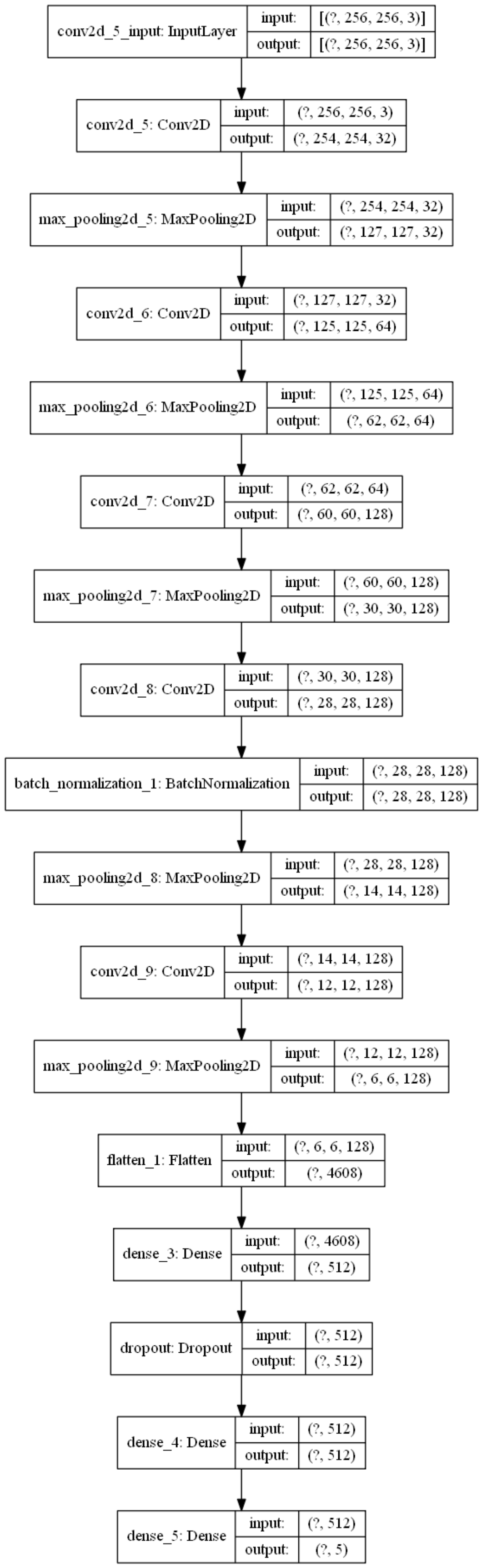
[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

VII. Appendix

A. *structure of the sequential CNN without dropout layer*


conv2d_input: InputLayer | input: [(?, 256, 256, 3)] | output: [(?, 256, 256, 3)]
conv2d: Conv2D | input: (?, 256, 256, 3) | output: (?, 254, 254, 32)
max_pooling2d: MaxPooling2D | input: (?, 254, 254, 32) | output: (?, 127, 127, 32)
conv2d_1: Conv2D | input: (?, 127, 127, 32) | output: (?, 125, 125, 64)
max_pooling2d_1: MaxPooling2D | input: (?, 125, 125, 64) | output: (?, 62, 62, 64)
conv2d_2: Conv2D | input: (?, 62, 62, 64) | output: (?, 60, 60, 128)
max_pooling2d_2: MaxPooling2D | input: (?, 60, 60, 128) | output: (?, 30, 30, 128)
conv2d_3: Conv2D | input: (?, 30, 30, 128) | output: (?, 28, 28, 128)
batch_normalization: BatchNormalization | input: (?, 28, 28, 128) | output: (?, 28, 28, 128)
max_pooling2d_3: MaxPooling2D | input: (?, 28, 28, 128) | output: (?, 14, 14, 128)
conv2d_4: Conv2D | input: (?, 14, 14, 128) | output: (?, 12, 12, 128)
max_pooling2d_4: MaxPooling2D | input: (?, 12, 12, 128) | output: (?, 6, 6, 128)
flatten: Flatten | input: (?, 6, 6, 128) | output: (?, 4608)
dense: Dense | input: (?, 4608) | output: (?, 512)
dense_1: Dense | input: (?, 512) | output: (?, 512)
dense_2: Dense | input: (?, 512) | output: (?, 5)

B. structure of the sequential CNN with a dropout layer


conv2d_5_input: InputLayer
input: [(?, 256, 256, 3)]
output: [(?, 256, 256, 3)]
conv2d_5: Conv2D
input: (?, 256, 256, 3)
output: (?, 254, 254, 32)
max_pooling2d_5: MaxPooling2D
input: (?, 254, 254, 32)
output: (?, 127, 127, 32)
conv2d_6: Conv2D
input: (?, 127, 127, 32)
output: (?, 125, 125, 64)
max_pooling2d_6: MaxPooling2D
input: (?, 125, 125, 64)
output: (?, 62, 62, 64)
conv2d_7: Conv2D
input: (?, 62, 62, 64)
output: (?, 60, 60, 128)
max_pooling2d_7: MaxPooling2D
input: (?, 60, 60, 128)
output: (?, 30, 30, 128)
conv2d_8: Conv2D
input: (?, 30, 30, 128)
output: (?, 28, 28, 128)
batch_normalization_1: BatchNormalization
input: (?, 28, 28, 128)
output: (?, 28, 28, 128)
max_pooling2d_8: MaxPooling2D
input: (?, 28, 28, 128)
output: (?, 14, 14, 128)
conv2d_9: Conv2D
input: (?, 14, 14, 128)
output: (?, 12, 12, 128)
max_pooling2d_9: MaxPooling2D
input: (?, 12, 12, 128)
output: (?, 6, 6, 128)
flatten_1: Flatten
input: (?, 6, 6, 128)
output: (?, 4608)
dense_3: Dense
input: (?, 4608)
output: (?, 512)
dropout: Dropout
input: (?, 512)
output: (?, 512)
dense_4: Dense
input: (?, 512)
output: (?, 512)
dense_5: Dense
input: (?, 512)
output: (?, 5)