****

**CSD361**

**Introduction to Machine Learning**

Project Report

**CASSAVA LEAF DISEASE CLASSIFICATION**

**DEVANSH GOEL 1910110132**

**DEVYANSH SEHGAL 1910110134**

**TANMAY JAIN 1910110416**

**Developing a Deep Learning Pipeline for Classifying Cassava Leaf Diseases**

An exploration into the components of an effective CNN model: from data extraction and cleaning to transfer learning and architectural design to hyperparameter tuning and more…



# Background and Motivation

One of Africa’s most crucial staple crops, the starchy cassava plant is the second-largest producer of carbohydrates on the entire continent, and, while this plant is known for its hearty nature and ability to withstand harsh environmental conditions, rampant disease outbreaks often threaten crop yields and pose a serious threat to the subsistence farmers who grow them. While over 80% of small, household farms in Sub-Saharan Africa grow this root, few have the ability to detect and mitigate the devastating effects of disease outbreaks with which they are regularly plagued.[2] At present, in order to assess whether one’s plants are stricken with disease, farmers must work with local government officials to deploy agricultural experts to inspect the plants in person. Unfortunately, this process is extremely labor intensive, slow, and inefficient which puts farmers at greater risk for losing larger portions of their harvest if they are indeed dealing with an outbreak amongst their plants. In order to help speed up this process and provide farmers with the best opportunity to save their crops

**Dataset**

The dataset we are provided with to create and optimize our network with contains 21,397 observations of cassava leaf images and their corresponding disease classifications. The five categories that the leaf images fall under are Cassava Bacterial Blight (CBB), Cassava Brown Streak Disease (CBSD), Cassava Green Mottle (CGM), Cassava Mosaic Disease (CMD), and Healthy. Some example images of these are shown below.

As can be seen from the images below, while there may be some distinct characteristics of these diseases to the trained eye, the traits indicative of illness are subtle and the ability to distinguish between these different symptomatic presentations and accurately diagnose the plants requires a level of expertise and skill not possessed by many.



Approach 1 :-

# Exploratory Data Analysis

As we begin to create a model which will allow for accurate and robust classification, we must first familiarize ourselves with the data at our disposal since the data is the foundation of a good model. Once we uploaded our data into Google Colab (an undertaking on its own), we were able to begin understanding the nature of the data which we would be using to train and base our model on. Our training set consisted of 21,397 pairs of cassava leaf images along with their corresponding disease classification.

One of the characteristics that was consistent for all of the images, however, was that their dimensions were all 800x600 (width by height) which made it more convenient to resize all of them in a ratio preserving manner without fear of distorting or warping any of the relationships between shapes and colors in the image.

**Drawbacks Of approach 1**

After analyzing our data using the value\_counts function, it became clear that the Cassava Mosaic Disease (CMD) was afflicting the vast majority of the plants depicted in the images, thus, in order to get a baseline score for our model, we decided to simply put the entire probability mass of our prediction function on that disease classification independent of the image provided to it. While this method is not particularly meaningful in terms of data analysis and does not even utilize the power of neural networks

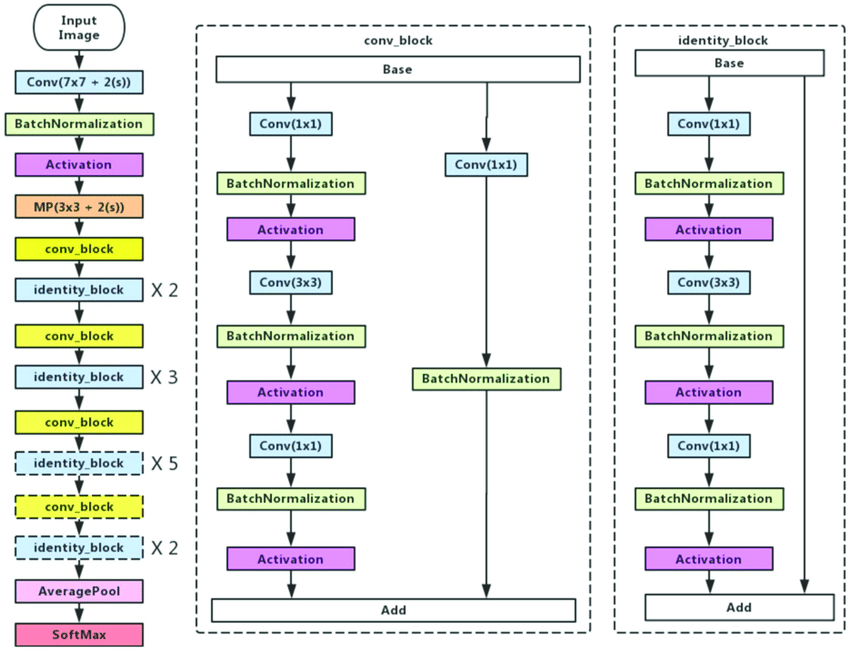
**Approach 2 :- CNN**

Our current goal is to create a CNN that will enable subsistence farmers in Sub-Saharan Africa to upload photos of their crops and find out whether or not their plants are healthy or diseased, and, if they are diseased, what they are stricken with. We left off having simply created a majority classifier model that always identified an image as being afflicted with Cassava Mosaic Disease (CMD). While this naive approach technically produced a test accuracy of roughly 60%, it was not a particularly meaningful diagnosis and does little to help the farmers who need rapid and accurate results in order to increase their yields and reduce unnecessary crop loss. In order to create a model that achieved this goal, we actually had to redo some of the data loading we had previously done in order to make our data pipeline more efficient. Only once the data was stored correctly could we begin the process of actually building our model and adjusting the architecture and hyperparameters to produce better results.

**RESNET50**

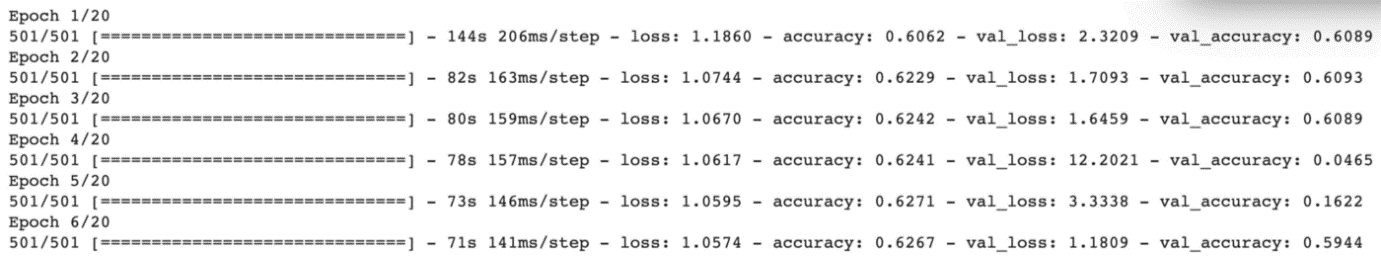
While uploading a base model is not necessarily required, a popular way to train a new model is to use a pre-trained model and perform some sort of transfer learning with it. After taking a deep dive into some of the best performing image classifier base models, we decided to use [ResNet 50](https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035)

Due to our limited RAM and disk space on Google Colab, we felt a 50 layer base model was large enough to emulate the complexity of our data but small enough to operate at reasonable speeds.



We started by importing ResNet50, freezing its layers, and adding 2 convolutional layers and several dense layers on top (the architecture will be discussed in detail in the following section).

Unfortunately, the capacity of our model was clearly too low to learn the complexities of the data set. Not only were the training loss and training accuracy starting to stagnate very early in the training process, but the validation loss and validation accuracy were extremely volatile which is a symptom of underfitting.



Training results with frozen base model

At this point it was clear that we would not be able to simply freeze all of the base layers and hope that our additional layers would provide enough flexibility in the model to appropriately fit and conform to our data, but, thankfully, ResNet50 has a low enough number of trainable parameters that training the entire model would not be too taxing on our system. Additionally, one of the benefits of freezing the base layers is that it is supposed to cut down on training time dramatically, but we actually found only a minor reduction in runtime that did not at all outweigh the devastating effects the freezing had on the model’s performance. All of this led us to the conclusion that we would have to leave the base model’s parameters trainable, and we were optimistic that this combined with a set of fully connected layers would allow our model to produce more desirable results.

**Architectual Design**

 there are only three fully connected layers between the base model and the output layer, the usage of batch normalization along with the solid pairing of ReLU activation and He normal kernel initialization makes it more powerful than it looks.

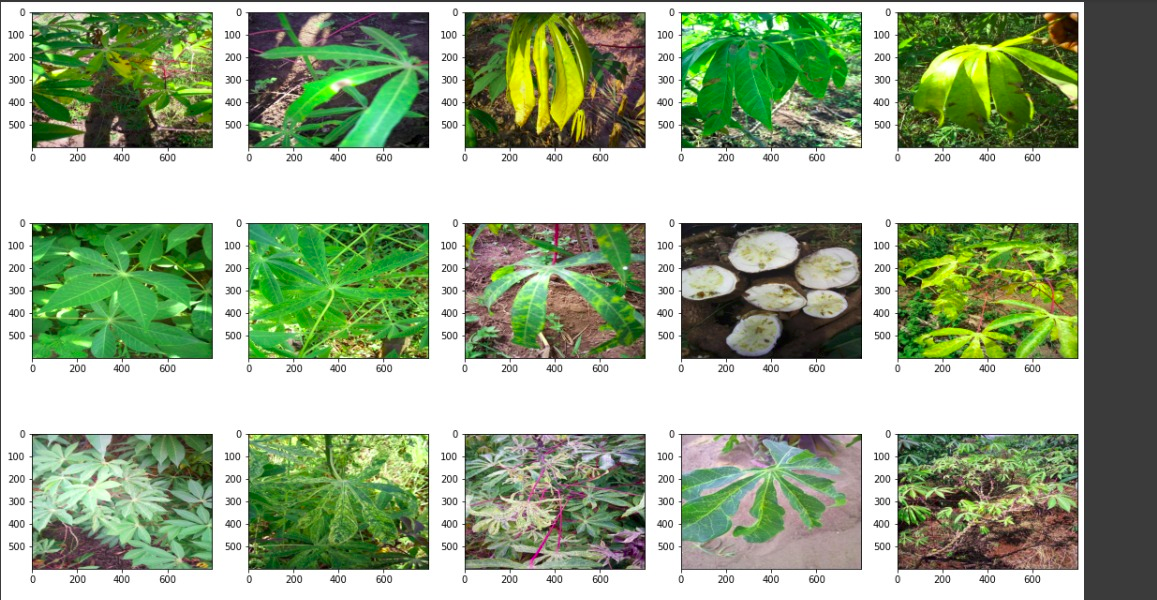
 We also included some light L2 regularization in these layers to help stem overfitting, but hyperparameter decisions such as this are discussed further in the next section. Overall, we now had a structure for our model and just had to find the ideal values for the hyperparameters that would determine how the training process would unfold.

# Hyperparameter Tuning

Having settled on an architecture we felt comfortable with fairly early on in the process, rather than improve our model’s performance through modifying hyperparameters that impacted the complexity, we focused on tuning those which influence the training process directly. Specifically, we endeavored to hone in on the best learning rate, batch size, and regularization values for our model. After reading multiple articles that emphasized the monumental importance of choosing a good learning rate, we decided to prioritize this hyperparameter above all others and make sure we had identified the optimal value for it prior to manipulating the values of the others.

Our strategy was to run our model with different learning rates being applied to the [Adam optimizer](https://ruder.io/optimizing-gradient-descent/index.html" \l "adam) for a smaller number of epochs and try to determine whether the validation accuracy and loss seemed to diverge from the training values, and how well they actually caused the model to perform. We started by trying to identify the upper and lower bounds which would cause the training process to unfold in an ideal manner, and, based on the results below, came to the conclusion that learning rates between 0.00001 and 0.0001 were ideal. Since we were including techniques such as batch normalization in our model and were compiling using the adaptive optimizer Adam, we felt as though a slightly more aggressive learning rate would be appropriate and help our model to converge on its optimal configuration more rapidly and efficiently.

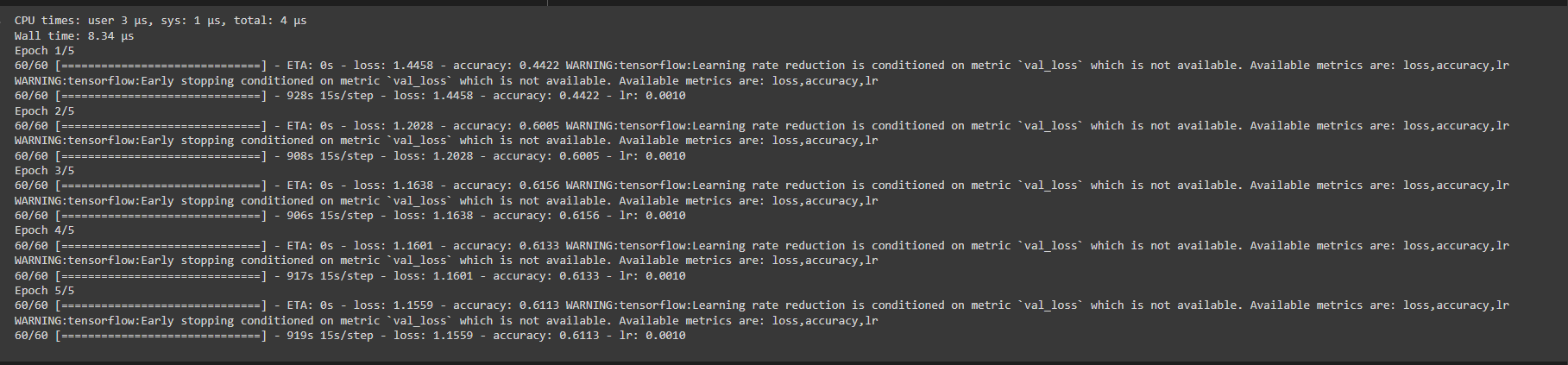
Having isolated 0.0001 as the ideal learning rate, we proceeded to test a variety of batch sizes to see whether increasing the value from the starting point of 16 to higher multiples such as 32, 48, 64, and 128 would cause our model to generalize better or reduce its ability to differentiate the various relationships between features and output classes since the variety within the batch might be too high to learn as many specific interactions.

Classified Images

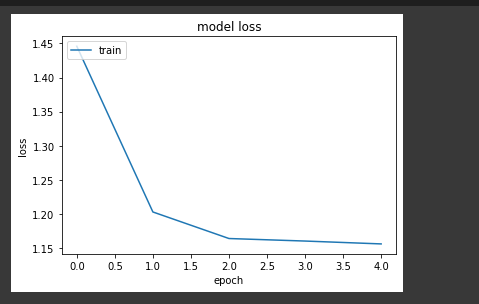
# 

**Model Performance**

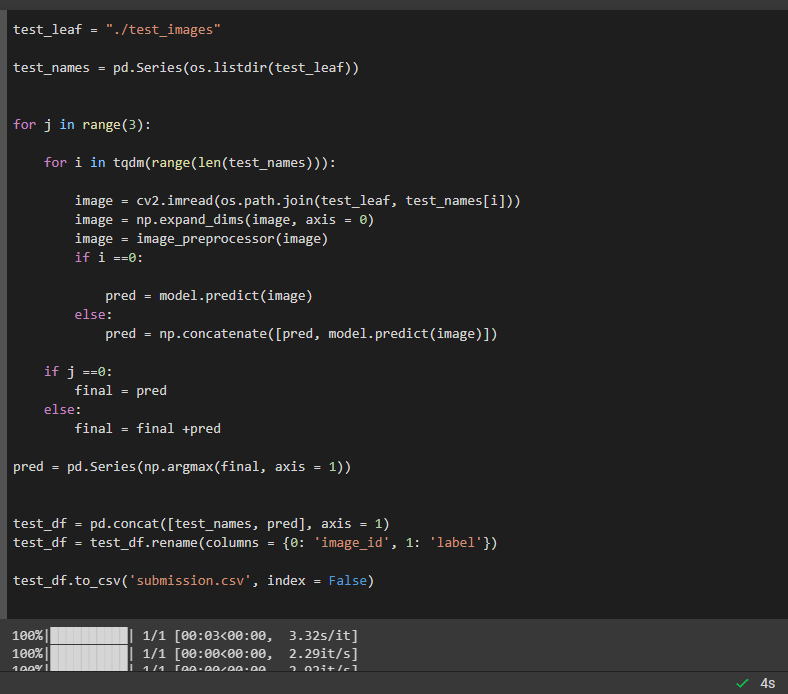
At its current stage, given the configuration described above, our model has been able to produce a consistent and reliable validation accuracy of roughly 80% and has produced the training graph depicted below which shows the validation loss tracking the training loss quite closely and also the validation accuracy steadily increasing over the epochs and not lagging too far behind the training accuracy, thus indicating only a minimal amount of overfitting. It would still be ideal to see the validation accuracy not plateau quite as early and the validation loss to continue on a downward trajectory for the full 20 epochs, but we believe there are still a few methods we can utilize to better achieve these desired results in the next iteration of our model.



In addition to the promising results shown in the plot above, our model has also performed well on the official Kaggle testing data set and achieved a score of over 81%. While there is still work to be done, this represents a tremendous step up from the 60% achieved by our baseline majority classifier model and moves us far closer to our goal of having a model that could actually be used in the field to help positively impact people’s lives through data. This is a decent spot to be in currently, but, as discussed in the next section, we continue to look ahead and consider how we can further improve our model.



Comparison with test image



# **Extra**

# We also tried to use the Efficient Net. So, Google recently published both a very exciting paper and source code for a newly designed CNN (convolutional neural network) called EfficientNet, it is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient.

# However, we were not able to go forward with that due to the time limitations as it takes a lot of time to fit. Still, we have attached the other ipynb file as well.

# **Future Work**

# Although we have made major improvements over the past few weeks, there are still a couple more features that we plan to include in our model and believe should result in even better model performance. Notably, the last two techniques we hope to implement are applying more extensive data augmentation and preprocessing methods to our images so that our model is being trained on a greater diversity of data that will hopefully allow it to generalize better, and using ensembling to combine the results of our current model with those of another in order to boost the performance even more. We specifically plan on exploring how using random flips, rotations, and color alterations will affect our model performance and whether another pretrained model such as CropNet will provide the additional information necessary to boost our model to the next level. Although we are still quite a few degrees below the threshold of 90% test accuracy necessary to be featured on the leader board, we have made significant strides since our previous post and firmly believe that there is still room for improvement through the use of a few additional methods such as those just mentioned. Overall, our model has come together quite nicely, and we have learned a lot through the process of developing it.

# **Bibliography**

# https://www.kaggle.com/code/tjain69/cassava-leaf-disease-using-resnet50/edit

* https://www.kaggle.com/code/tjain69/cassava-leaf-disease-using-resnet50/edit
* https://www.youtube.com/watch?v=bezp-iv1RGk
* https://www.kaggle.com/c/cassava-leaf-disease-classification
* https://ieeexplore.ieee.org/document/9615488
* The notes and Slides used in class.

**Thank You Note**

**We would like to thank Professor Harish Chandra Karnick to teach us Introduction to Machie Learning even in online mode. We would also like to thank Prakhar Rathi, the teaching assistant of the course for helping us and clearing our doubts.**