Group 2, Machine Learning Project Report

Detecting Cassava Leaf Diseases Using Different Deep Learning Architectures

Serghei Mihaliov M.Bahadir Kucuk
Athanasios Serntedakis Pelle Drijver Mehmet Cetin

Abstract

In this project, we benchmark the performance of the following deep learning architectures: EfficientNetB0, ResNet50, DenseNet169 and MobileNetV3 (small versions of state-of-the-art convolutional network architectures) on the Cassava Leaf Disease Classification challenge. The dataset for this challenge consists of 21,367 labeled photos of Cassava leaves taken via farmers' phones, distributed over 5 classes: 1 healthy and 4 diseases. The goal is to see how well architectures' performance on ImageNet transfers to this challenge. Finally, we evaluate the performance of the ensemble of the 4 models, with the goal of achieving a high score on the challenge.

1 Introduction

Deep Convolutional Neural Networks (DCNNs) [12] have developed at a high pace during the last decade, leading to major breakthroughs in the field of computer vision, including image classification. There is now an abundance of DCNN architectures and their combinations, leveraging visual information in conceptually different and innovative ways. The shared elements among them are the use of alternating convolution and pooling layers, with fully-connected layers at the output of the network. Dropout and batch normalization are common [9]. The principal differences between architectures are the arrangements, connectivity and grouping of layers. Judging by the most popular datasets for benchmarking image classification architectures, ImageNet [3], CIFAR-10 and CIFAR-100 [11], such variations can play a crucial role. Their impact will be discussed in detail for the concrete architectures that we benchmark.

1.1 Hypothesis

ImageNet, one of the most popular datasets used as a benchmark for DCNN architectures, and is often used as an indication of their quality. Our hypothesis is that the performance of DCNN architectures on ImageNet would not translate into a similar relative performance to the Cassava Leaf Disease Classification challenge, leading to models with similar performances on ImageNet to have dissimilar performances on CLD, due to differences in their working principles. Moreover, we hypothesize that using an ensemble of models based on such architectures, when trained on the Cassava Leaf Disease dataset, would significantly improve the accuracy of each individual model.

1.2 Data description

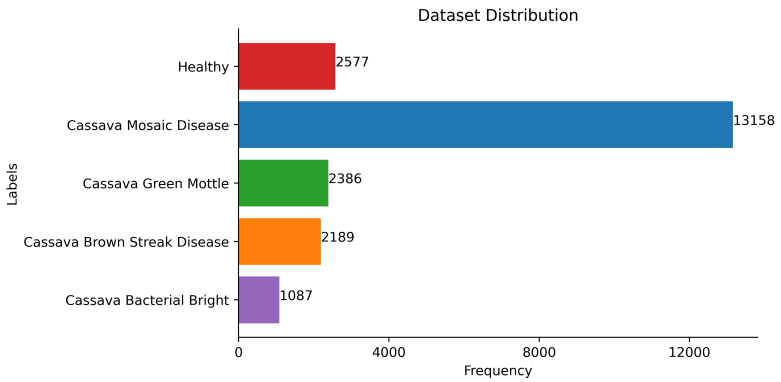


Figure 1: The distribution of the training dataset

The dataset we used contains 21,367 labelled images [13], each with 600 pixels width and 800 pixels height. The images are distributed in five different classes, one for healthy Cassava leaves, and four for the other different types of diseases. The labels and class distribution can be found in Figure 1.

We can observe that dataset is clearly imbalanced, as there is an unequal distribution of classes. The Cassava Mosaic Disease has a frequency of 13,158 images, which constitutes the majority of the dataset. Later in this paper we will discuss how we managed to prevent the class imbalance, by augmenting images.

2 Architecture selection

The main selection criteria for architectures were 1) accuracy on the ImageNet challenge [3] [14], 2) model compactness in terms of number of parameters, 3) similar performance on ImageNet of the selected models, so that performance differences on the Cassava challenge, if present, would be meaningful.

To satisfy criteria 1, 2, we selected the distinct architectures used in the top 300 models submitted to the ImageNet challenge, and filtered out those with over 50M parameters, so that each model could be optimally trained in a few hours on a consumer computer. In Table 1 we outline the architectures identified to match our criteria. Finally, we selected the following group of models with a similar performance, between 75.2% and 77.1%: MobileNet V3-Large 1.0, ResNet-50, DenseNet-169 and EfficientNet-B0. Another group that we considered were Xception, Inception-v3, ResNet-101, DenseNet-264 and EfficientNet-B1 with Top-1 accuracy between 77.9% and 79.1%, but we opted for the more compact models.

Model	Top-1 acc.	# Parameters (M)	ImageNet rank	Year
MobileNet V3-Large 1.0 [6]	75.20%	5.40	264	2019
ResNet-50 [4]	76.00%	26.00	228	2015
ResNet-101 [4]	78.25%	40	212	2015
DenseNet-169 [8]	76.20%	14.00	247	2016
DenseNet-264 [8]	77.90%	34.00	218	2016
Inception-v3	78.80%	24.00	194	2015
EfficientNet-B0 [21]	77.10%	5.30	244	2019
EfficientNet-B1 [21]	79.10%	7.80	191	2019
EfficientNet-B2 [21]	80.10%	9.20	164	2019
EfficientNet-B3 [21]	81.60%	12.00	136	2019
EfficientNet-B4 [21]	82.90%	19.00	101	2019
Xception [1]	79.00%	23.00	182	2016

Table 1: Overview of architectures considered for benchmarking, based on performance and size. Highlighted in green are the selected models.

2.1 ResNet

A residual network [4](ResNet) is a deep-learning framework for image classification that is capable of handling networks with many stacked layers. Research has shown that the depth of a network plays an important role in terms of accuracy for the task of image classification [20]. The main problem of models with multiple stacked layers is that the model starts to degrade after some point, or exhibit vanishing gradients, resulting in a high training error [19]. A residual network tries to tackle this problem by introducing a new building mechanism, the residual building block.

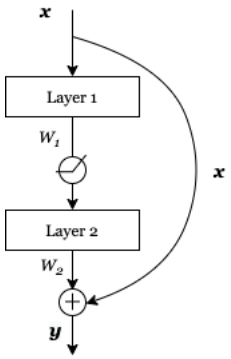


Figure 2: A residual building block

Residual Building Block

A residual block(Figure 3) consists of the usual stacked layers with a shortcut connection from the input of the layer 1 to to output of the layer 2. Formally, this building block can be defined as follows:

$$y = W_2 \cdot \sigma(W_1 \cdot x) + x$$

Where $F = W_2 \cdot \sigma(W_1 \cdot x)$ denotes the residual function.
The residual block is flexible in terms of skipping layers, so the residual function F may contain more that 2 layers. Moreover, the shortcut connection can incorporate a transition

from different input and output dimensions by a linear projection W_s:

$$y = F(x) + W_s \cdot x$$

To do backpropagation with this construction, we need to work out the gradient of the loss(L):

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial (F + x)}{\partial x} = \frac{\partial L}{\partial y} \cdot \left(\frac{\partial x}{\partial x} + \frac{\partial F}{\partial x} \right) = \frac{\partial L}{\partial y} \cdot \left(1 + \frac{\partial F}{\partial x} \right) \tag{1}$$

From (1) we can observe that the identity mapping of the shortcut connection provides another route for the gradient to flow that increases its signal during backpropagation to prevent vanishing gradients. With this simple addition of the shortcut connection the network can reach deeper depths of stacking without the problem of degradation or vanishing gradients compared to plain deep networks as presented in the original paper of the architecture [4].

2.2 MobileNet

The MobileNet architecture is originally based on the concept of using a streamlined architecture that uses depthwise separable convolutions to build lightweight deep neural networks [7]. Since we were limited in time and resources ourselves, this architecture particularly stood out to us. The architecture has become increasingly more popular due to its strong performance while matching the resource restrictions (latency, size) of a mobile application. Over time, MobileNet has subsequently improved through novel architecture advances, until it reached its current generation, MobileNetV3 [6]. Because of the large number of advances in resource- and time efficiency of the architecture, we will provide a quick summary of one of the efficient building blocks used to construct MobileNets: depthwise separable convolutions.

Depthwise separable convolutions

The depthwise separable convolutions have been part of the architecture since MobileNetV1 as an efficient replacement for traditional convolution layers [7]. Rather than creating a deep convolution with many channels, the depthwise separable convolutions factorize a standard convolution into two other components: a depthwise convolution and a 1×1 convolution called a pointwise convolution that creates a linear combination of the output of the depthwise layer. This has the effect of drastically reducing computation time and model size, while having a slight reduction in accuracy. Its ability to reduce the dimensionality of a layer thus reducing the dimensionality of the operating space has been further exploited in the next generations (i.e. the inverted residual with a linear bottleneck) [17].

2.3 DenseNet

Dense Convolutional Network (DenseNet) [8] , introduced in 2016, is a convolutional network that leverages the performance improvements from closer connections between the layers close to the input and the layers close to the output that allow for feature reuse. In contrast to traditional convolutional networks, DenseNet consists of blocks of layers, with each layer in the block connected to all its predecessors in a feed-forward fashion. Transition layers between blocks perform downsampling via convolution and pooling. The inputs from the previous layers are concatenated (input alongside each other), and not summed like in ResNets.

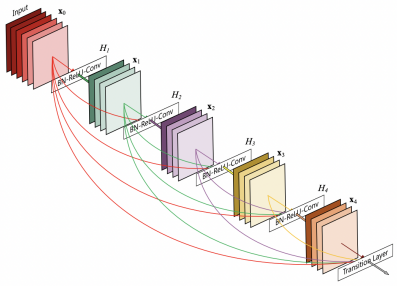


Figure 3: A DenseNet block [8]

DenseNet is parametrized by growth rate k, which determines the number of channels per input from a predecessor later, the presence or absence of a 'bottleneck' - 1x1 convolution layer before each 3x3 convolution to reduce the number of feature-maps passed, and compression rate θ, which determines the ratio of feature-maps retained after the transition layer.

We used the DenseNet169 variant, developed for ImageNet, for benchmarking. It uses bottleneck, compression with experiments on ImageNet, we use a DenseNet-BC $\theta = 0.5$, growth rate $k = 32$, 4 dense blocks.

The advantages of DenseNet claimed by the authors are the alleviation of the vanishing-gradient problem, increasing feature propagation and feature reuse, as well as reducing the number of parameters. Moreover, the paper indicates that the model has no difficulties scaling to hundreds of layers without performance degradation or overfitting. Somewhat counterintuitively, passing feature-maps from previous layers results in fewer parameters required for comparable accuracy on challenges like CIFAR-10, CIFAR-100 and ImageNet.

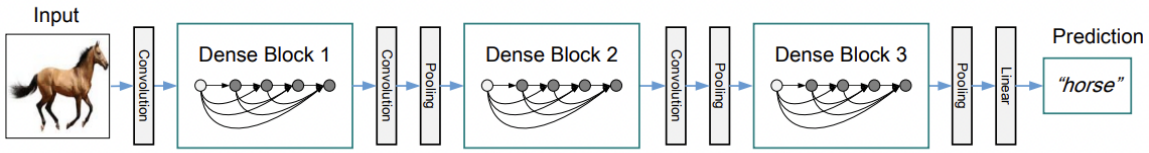


Figure 4: The DenseNet architecture [8]

2.4 EfficientNet

EfficientNet is a convolutional network (ConvNet) that uses a model scaling method by balancing the width, depth, and resolution of the network. To balance the dimensions, Tan et al. propose a simple and highly effective compound scaling method, which balances the scaling of width, depth and resolution in a more principled way, while maintaining model efficiency. Moreover, scaling dimensions is difficult because optimal width, depth, and resolution values depend on each other. Due to this difficulty, conventional methods mostly scale CovNets into one of these dimensions: depth, width, or resolution.

Depth

Scaling network depth is used by many ConvNets [5] because ConvNet's with deeper networks can capture richer and more complex features, and generalize well on new tasks. Deeper ConvNets, however, are more difficult to train due to the vanishing gradient problem [22].

Width

Scaling network width is mostly used for small size models [7]. The reason for increasing the width of a CovNet is to capture more fine-grained features. Nonetheless, increasing only the width of the ConvNet leads to shallow networks that tend to have hard times while capturing higher-level features. According to Tan et al. the accuracy quickly stops increasing when networks are shallow with extreme width.

Resolution

In ConvNets, input images with higher resolution values can capture more fine-grained patterns. Tan et al. advocate that higher resolutions improve model accuracy, but the accuracy gain diminishes for extremely high resolutions.

The advantage of EfficientNet is that when scaling the network, Efficientnet uses compound scaling and balances all the dimensions of the network and obtains better accuracy and efficiency [21]. We used EfficientNet to inspect how EfficientNet achieves the highest accuracy among all the Convolutional network architectures used in this paper.

3 Ensembling

In order to achieve an accuracy as high as possible on the Kaggle hidden test-set, we applied a simple form of ensembling that combines the predictions of all models we've optimized and trained. This way, we can attack the learning problem with different types of models which are capable to learn some part of the problem, but not the whole space of the problem. Our stacked model takes the predictions of the others as input, adds a neural network as a combiner with a single hidden layer of 32 nodes on top, and computes the output with a softmax over five output nodes. Due to our stacked model's high sensitivity to overfitting, we decided to train it on precisely one epoch of the data.

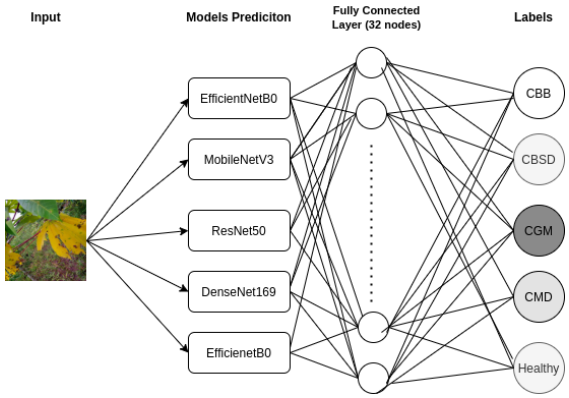


Figure 5: Ensembled model

4 Methodology

4.1 Data Preprocessing and Augmentation

As we mentioned before, our dataset suffers from a class imbalance. Class imbalance in training data can hurt a model's performance on real data by biasing it to predict instances as the majority class. Data augmentation allows us to extract more data from the dataset by applying techniques such as geometric and color transformations. Moreover, it can be used to alleviate class imbalance when used to oversample [18]. To prevent class imbalance, we generated additional augmented images to obtain a dataset in which the frequency of all classes are equally distributed.

The original images were augmented by applying a number of naive geometrical preprocessing operations. First, we resized the images to the dimensions 224x224. Sequentially, we added a random rotation with a factor between -0.2 and 0.2 and applied a random zoom with a factor between 0% and +30%. This is a simple approach, but it meant we should avoid the pitfall of overfitting on the minority class instances, due to the relative similarity of the augmented images, compared to using GANs or neural style transfer. By augmenting the original images from the datasets until the frequency of each class is equal to the frequency of the most common class, we obtained a new uniform distribution of the classes.

An example of the augmented images obtained from an original image is presented in Figure 6.

(a) Original Image

(b) Augmentations

Figure 6: Original & augmented images of a Cassava Bacterial Blight labelled leaf

4.2 Splitting data

In to evaluate the architectures' performance on the classification of cassava leaves, we've carried out a train-validation-test split. This allowed us to evaluate each model's performance on a set of class-balanced instances it has never seen before during training. The validation and test sets are both created by removing 200 instances from each class from the original augmented data. The validation data is used when optimizing each model with the right hyperparameters, and the test data is used for our final architecture evaluation.

4.3 Benchmarking

In addition to Kaggle's hidden test-set of the Cassava Leaf Disease Classification challenge, we have created our own test-set. The main reason behind this is because Kaggle does provide any other evaluation metric on their hidden set other than accuracy. However, our research when comparing two architectures involves other

metrics such as the confusion matrix, which we would also like to analyze for each model. This led us to create our own separate test set for which we report our results in the result section.

While training our models for each architecture with the optimal hyperparameters found using tuning, we perform a number of logging procedures that keep track of a model's performance. These metrics include: training speed, training accuracy, test accuracy, and the confusion matrix. This allows us to not only to draw conclusions on the final model, but also the performance during training (time to converge, alternation of generalization gap over time, etc.).

Due to computational resource limitations, we opted to train for 30 epochs, so that training does not exceed 9 hours. Moreover, as discussed in the next section, the smallest version of architectures are selected, such that 30 epochs are sufficient for convergence.

4.4 Running environment

We've used Google Colaboratory to run all of our scripts for data augmentation, tuning, training, and evaluation to ensure that we use the same hardware for training across the team. Colaboratory provides computational resources similar to those of a top-of-the-line consumer laptop. It runs 2-core Xeon 2.2GHz CPUs, 13GB RAM. The GPUs available in Colab often include Nvidia K80s, T4s, P4s and P100s and are rotated for availability reasons.

4.5 Hyperparameters

4.5.1 Hyperparameter choice

Based on our observations of the best performing models on the CLD Kaggle challenge, we observed that 30 epochs of training and an image size of 224x224x3 was sufficient for convergence and gold-level score, although larger image sizes were more often used. GPU memory limits coupled with image size capped the batch size hyperparameter at 32, which was also a common value among gold-level submissions.

4.5.2 Hyperparameter tuning

To ensure that the training process is fair, we used tuned the learning rate and label smoothing hyperparameters for each model using the efficient Hyperband hyperparameter optimization algorithm [15]. We used the training dataset for training and validation dataset for measuring hyperparameter performance, with validation accuracy being the optimization objective. We allowed at most 15 epochs per combination of hyperparameters for optimization, and used a reduction factor of 3 for the number of epochs and models for each bracket. Moreover, we used learning rate reduction during tuning, same as used during training.

	Learning Rate	Label Smoothing
ResNet50	0.02359	$3.815 \cdot 10^{-9}$
DenseNet169	0.02292	$1.019 \cdot 10^{-7}$
EfficientNetB0	0.6720	$1.040 \cdot 10^{-6}$
MobileNetV3	0.1795	$6.774 \cdot 10^{-5}$

Table 2: Learning rate & label smoothing hyperparameters overview per model

4.6 Model implementation

Since all 4 architectures are state-of-the-art and relatively popular, we were able to use their reference implementations directly from the Keras library [2]. This is important both from the point of view of ease of implementation and reproducibility. To clearly capture the performance of the architectures, we initialized the weights randomly, instead of using ImageNet weights. Moreover, for the final dense layer, we used 5 units instead of 1000, since there are 5 classes in this dataset. The input shape we used was identical to that of the ImagenNet challenge. For all models, we used the softmax classifier activation.

We used Adam [10] to optimize categorical cross-entropy loss, with learning rate reduction on plateau and label smoothing.

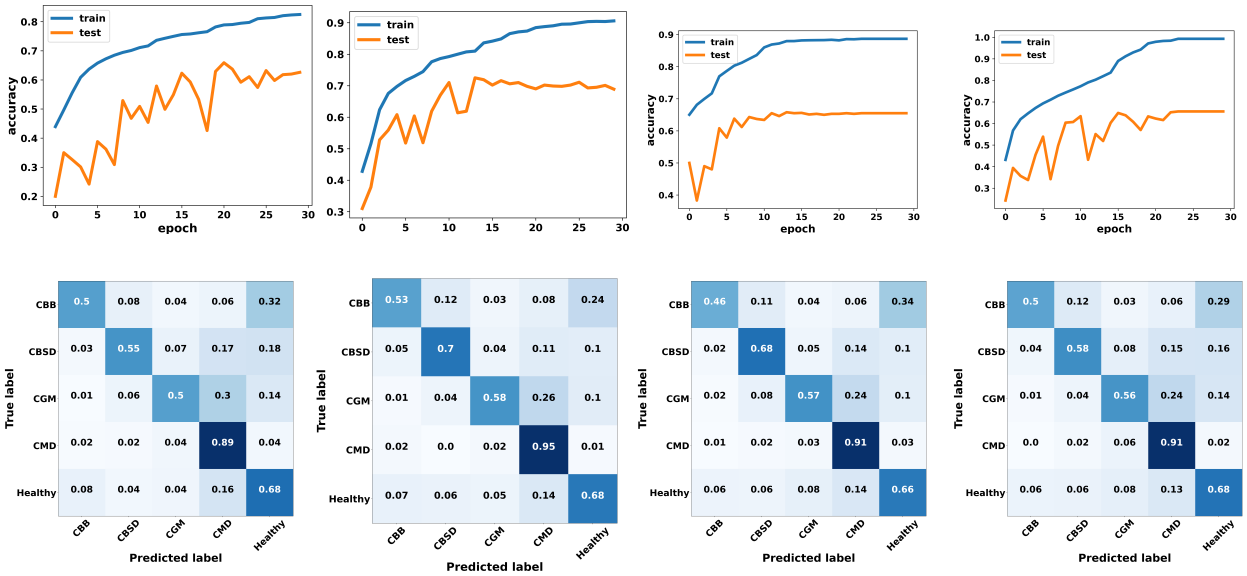
4.7 Learning rate reduction on plateau

To optimally train the model, we used a technique called learning rate reduction on plateau. This allows faster training during first epochs, then diminishing the learning rate once validation loss begins to plateau, allowing the model to converge. We used a patience parameter of 3 and a reduction factor of 0.2, meaning that the learning rate would decrease to 0.2 of its value after 3 epochs of no improvement on validation loss. We set a minimum learning rate of 0.0001 to prevent overfitting, compared to the initial learning rates ranging from 0.02292 to 0.6720.

4.8 Evaluation

To evaluate the performance of the models, we measured the accuracy on the test dataset withheld from available data and the accuracy on the hidden dataset for this challenge, by submitting the models to Kaggle. Moreover, we recorded the training and validation accuracy, loss and learning rate for every epoch, and the confusion matrix for every 5 epochs.

5 Results



(a) MobileNetV3 (b) EfficientNetB0 (c) DenseNet169 (d) ResNet50

Figure 7: Overview of the accuracy per epoch and confusion matrix for each model.

Table 3: Overview of how each network performed.

Models	Accuracy	Kaggle Accuracy
Resnet50	0.656	0.7879
DenseNet	0.654	0.7748
EfficientNetB0	0.689	0.8001
MobileNetV3	0.645	0.7128
Ensembled Model	0.69	0.8315

6 Discussion

6.1 Results overview

As previously explained, we evaluated the performance of the models as measured the accuracy on the withheld test dataset, the hidden test dataset, the confusion matrices and the training process, and we made the following observations. *Observation 1: The accuracies on the Kaggle test set are higher and have a higher variance compared to the withheld test set.* The most likely explanation is that the Kaggle hidden test set might be

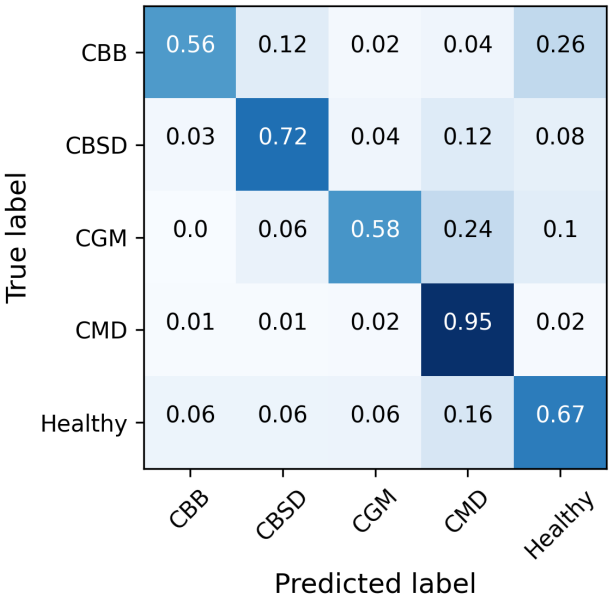


Figure 8: Ensembled model

imbalanced towards class 3, just like the public dataset, which results in our models to perform better on it than on our own test set withheld from the public dataset, because, as it can be seen from the confusion matrices, models have a high recall for class 3. Generally, at this point we can only speculate why this happened because the dataset remains hidden even after the submission, and therefore, there is no way to know and fully understand what caused this discrepancy between the local and the Kaggle's test data.

Observation 2: The distribution of accuracies of the models on the Cassava Leaf Disease Classification challenge is consistent with the performance of the corresponding models on the ImageNet dataset for the withheld dataset, but the variance is significantly larger on the Kaggle test set. As can be seen from the final results table, accuracies are clustered between 64.5% and 68.9% on withheld test set, and between 71.3% and 80.1% on the Kaggle test set, with ranking of models being similar: MobileNetV3 as the bottom performer, Resnet50 and DenseNet169 on par, and EfficientNetB0 as the top performer. The performance of the better architectures was better on the Kaggle test set, creating a larger gap, inconsistent with the distribution on ImageNet.

Observation 3: The ensembling model does outperform base models. Compared to the best performing model, EfficientNetB0, with 68.9% accuracy, the ensembled model has a 69.0% accuracy, which is not significant. In the case of the Kaggle test set, the accuracies are 80.0% and 83.2% respectively. This is likely due to models having similar biases for the images in the test set, as judged from the confusion matrices, whereas on the Kaggle test set, the models provide a distinct insights, giving the ensembling method an edge. Another possibility is that the test set from Kaggle is not balanced.

Observation 4: Predictions are more accurate for the class that was originally the majority class, even after data augmentation. Judging by the confusion matrices, all models had a high recall on the CMD class. This is not due to class imbalance, since the dataset had been balanced during preprocessing and augmentation. We infer that this is due to the class having a much higher number of non-augmented, highly-informative 'real' images. Thus, we hypothesize that using a dataset with a similar number of 'real' images of each class, the models could provide a very good solution to the problem at hand, with accuracies similar to those displayed on ImageNet. Another solution would have been the use of a loss function taking into account the disbalance of the dataset, such as the focal loss [16].

Observation 5: Training accuracy keeps rising during the whole process of training, while validation accuracy stagnates around epoch 15. This is usually an indicator that the model is overfitting, and training should be stopped. However, the scores on the hidden Kaggle dataset were consistently and significantly higher for models trained for 30 epochs than for models trained for 15 epoch, which is something that should be investigated in future work.

Observation 6: The accuracy of the models trained is low compared to the gold, silver, bronze results (top 390 out of 3900) on Kaggle, which have accuracies between 0.8978 and 0.9132, even though the top scorers used similar architectures, such as ResNet and EfficientNet, as well as basic ensembling techniques, such as averaging the outputs of the classifier level. We think that this is due to, primarily, data augmentation causing

more harm than good, resulting in oversampling of augmented images too similar to the original ones. Secondly, we used the most compact models, whereas larger models were used to achieve a high score on this challenge, e.g. EfficientNetB3.

Observation 7: Learning rate reduction on plateau consistently improved training and validation accuracy, as can be seen from the corresponding spikes on the accuracy graphs during training.

7 Conclusion

In this project, we benchmarked 4 different DCNN architectures, ResNet50, MobileNetV3, DenseNet169, and EfficientNetB0, on the Cassava Leaf Disease Classification dataset [13], to see how well their performance on ImageNet transferred to this dataset. The distribution of accuracies suggests that the relative performances of these architectures are very similar between the ImageNet and the CLDC challenge, with the ranking maintained, accuracies being similarly clustered on the withheld dataset, although more with more variance on the hidden test for CLDC on Kaggle. The ensembling, which we predicted would greatly improve performance, did not significantly exceed the best performing model.

Overall, the experiment process was fair to all models, but could be improved to increase the overall performance. As directions for future work, we suggest, primarily, to better handle class imbalance and data preprocessing. Data augmentation via rotation and zooming balanced the classes, bud did not produce augmented images that would contribute as significant information as the original ones. Instead of categorical cross-entropy loss, focal loss could be used to decrease the weight of the better predicted class, which would be the majority class. Secondly, more insight should be gained into the model performance, by visualizing the feature maps learned, better detecting and handling overfitting, as well as tuning deeper hyperparameters.

References

[1] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[2] F. Chollet et al. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k*, 7(8):T1, 2015.

[3] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[6] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for mobilenetv3, 2019.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[8] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[9] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, 2020.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[11] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[13] M. U. A. Lab. Cassava Leaf Disease Classification. `https://www.kaggle.com/c/cassava-leaf-disease-classification`, 2021. [Online; accessed 19-March-2021].

[14] F.-F. Li. Image Classification on Imagenet. https://paperswithcode.com/sota/image-classification-on-imagenet, 2021. [Online; accessed 26-March-2021].

[15] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[16] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection, 2018.

[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. 2018.

[18] C. Shorten and T. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48, 2019.

[19] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks, 2015.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014.

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

[22] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Appendix

Source code on Github: https://github.com/SergheiMihailov/ml-project-cassava