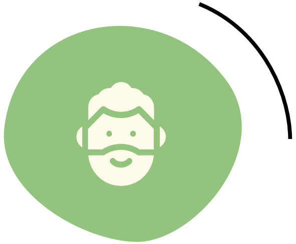
Cassava Leaf Classification

Mathematical Modelling Course, Applied Computational Intelligence MSc
Babeș-Bolyai University, Cluj-Napoca, Romania

Team members

Teofana Enăchioiu

MSc Student



Alex Adăscăliței

MSc Student

Ligia Novăcean

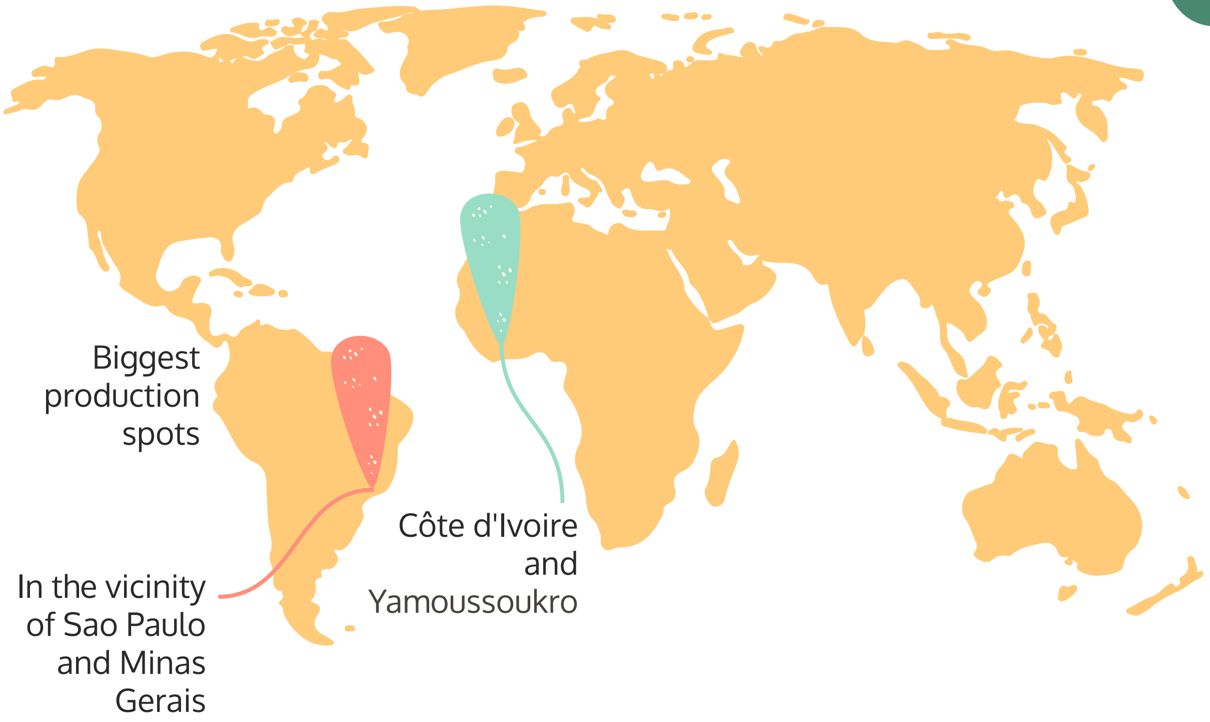
MSc Student

Github Repository

cassava-math-ubb

The final results are contained in the same organization, on Github, in cassava-math-ubb/end-results

Problem statement and importance



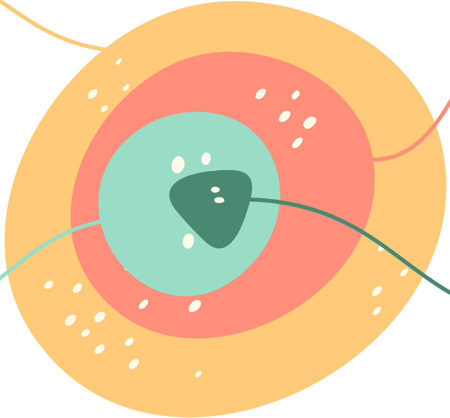
We had to classify each cassava image into four disease categories or healthy leaf. With our help, farmers may be able to quickly identify diseased plants, potentially saving their crops before they inflict irreparable damage. Important for human nutrition, usually boiled to avoid their toxicity, being the sugar beets the runner ups. Also, they are used in laundry products, ethanol biofuel, and alcoholic beverages.

Mosaic

13.158 images, a virus transmitted by whiteflies - no treatments

Green Mottle

2.386 images, a virus transmitted by nematodes, with distinctive yellow spots, green patterns - usually they recover



Brown Streak

2.189 images, presents necrotic vein banding and yellow patches, followed by necrosis on tubes and stems

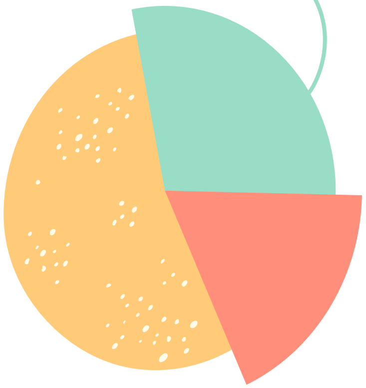
Bacterial Blight

1.087 images, water-soaked spots on the lower side of the leaves

Data labeled by the National Crops Resources Research Institute (NaCRRI) in collaboration with the AI lab at Makerere University, Kampala.

Public test data semantics distribution.

Although this **data** set contains 21397 images, it is highly imbalanced.



61.49%	60.2%	**Mosaic**
12.04%	14.1%	Healthy
11.15%	10.3%	Green Motle
10.23%	10.6%	Brown Streak
5.08%	4.8%	Bacterial Blight

Development

Found disease characteristics, ensured labels correctness, and formed a team on Kaggle and Github.

Problem statement

Started with ViT and Efficient Net B0, then cross-validation and optimizations

Developed the initial models

Moved training from local machines to the University's High Performance Computing Center, and outsourced GPUs

Training challenges

Development

Made multiple submissions on Kaggle with different models, such as EffNet. B0 to B3, advancing in the ranking

Final results

Refactored the code base, documented the progress and discussed the final results.

Thoughts and discussions

Ranked 1895/3510

In top 54%

Approaches

Similarity Metrics

In order for us to decide whether we should approach this challenge with a classical image-processing algorithm or a deep learning model, we made use of similarity metrics in a Monte Carlo study. For this task we used the Learned Perceptual Image Patch Similarity (LPIPS), peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) similarity measurements, as they are the most used ones.

EfficientNets family

EfficientNets offer a scaling alternative to increase the width, depth, and image resolution by a compound model scaling obtained through scaling all dimensions at once. In general, this scaling improves both the accuracy and efficiency of the models by reducing parameter size and FLOPS. In using this family of models we also took advantage of the transfer learning from ImageNet.

ViT

The Vision Transformer is an image classification model relying on a Transformer-like architecture applied on sequences of image patches. On the ImageNet dataset, the 2 largest models in the ViT family obtain top 10 results. Our initial plan was to experiment with this model, but even the smaller one (Vit-B/16) required an very long training time. Having around 86M parameters, experimenting with this promising and recent architecture was not feasible given our computing resources.

Pre-processing by means of similarity

Running the following experimental setup showed us that there is a remarkable difference between the images contained in the dataset, through metrics such as peak signal-to-noise ratio (PSNR) and structural similarity index measurement (SSIM). Given the circumstances, we decided to proceed with a recent and highly effective deep learning model instead of a classic data image processing algorithm. The Learned Perceptual Image Patch Similarity (LPIPS) metric gives more conservative feedback, but it never disagrees with our decision.

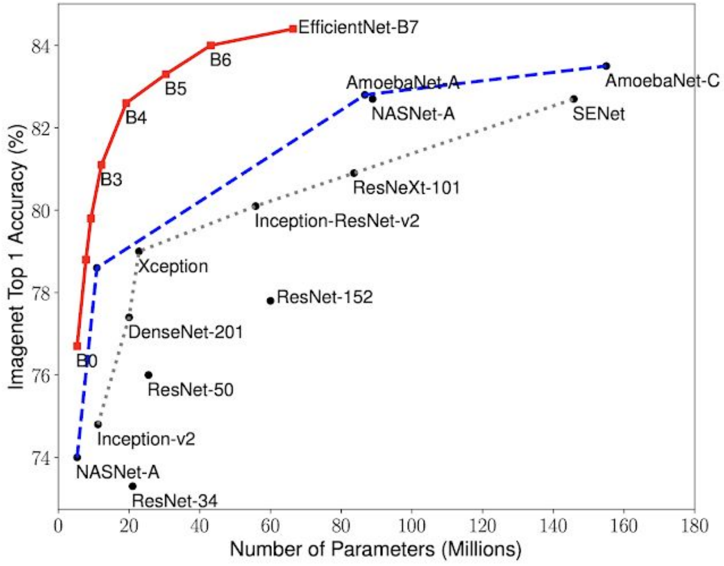
Computing such metrics requires a one-vs-all approach, yet 21397^2 exceeds a waiting time we could afford. One solution is to run a Monte Carlo study. The main purpose of simulations is estimating quantities whose direct computation is complicated, expensive, or time consuming.

```
Input: number of images loaded := 100, experiment sample size := 10, number of simulations := 10
Runtime:             LPIPS               PSNR                 SSIM
Simulation #1: 0.6593322157859802, 9.47935962677002, 0.09285126626491547
Simulation #2: 0.6850652098655701, 9.902604103088379, 0.08222384750843048
Simulation #3: 0.6983931660652161, 9.53113079071045, 0.0811685174703598
Simulation #4: 0.6989937424659729, 9.27914810180664, 0.09771811217069626
Simulation #5: 0.6332558989524841, 9.030553817749023, 0.05319840461015701
Simulation #6: 0.6239102482795715, 10.159566879272461, 0.06416226178407669
Simulation #7: 0.6591953635215759, 9.76210880279541, 0.07119724154472351
Simulation #8: 0.6343691945075989, 9.308271408081055, 0.07333287596702576
Simulation #9: 0.658652126789093, 9.396651268005371, 0.09231053292751312
Simulation #10: 0.6619075536727905, 10.314115524291992, 0.08350417762994766
Overview:
Metric        Value
--------  ---------
LPIPS     0.661307
PSNR      9.61635
SSIM      0.0791667
```

Notes:
- N set: Limit IO operations / Fit images in a ordinary computer RAM (one vs all with all would not fit 8GB RAM)
- the T simulations governs the Monte Carlo methodology
- K images: an arbitrarily decided amount of random images for one trail to become relevant

EfficientNets

One of the key issues in designing CNNs is model scaling i.e deciding how to increase the model size so as to provide better accuracy. In EfficientNets family, this problem is solved by scaling all dimensions of network width, depth, and resolution, using a global scaling factor.



These models have between 4M and 66M parameters, which is significantly less compared to other existing CNNs. The part that contributes to improving the efficiency is inherited from mobile devices networks and consists of an inverted bottleneck convolution (MBConv), which forms a shortcut connection between the beginning and end of a convolutional block.

We ran several experiments using EficientNet B0, B1, B2, and B3 and learning transfer from ImageNet and the highest accuracy we got on our dataset is 89%.

89.0

Top score: 91.1

Best model

Score obtained with the Efficient Net B3, 2.1% diff. up to the top submission

Architecture	Validation accuracy	Public test set score	Input size	Learning rate	Optimizer	Epochs	Notes	Training resources	Time
EfficientNet B3	0.887	**0.89**	512, 512	3.00E-04	Adam	40	• Test Time Augmentation: 5 steps • Loss: Label Smoothing Cross-Entropy (0.2) • Train augmentation: random resize and crop, horizontal flip, color jitter (Brightness, Contrast, Saturation: 0.25) • Test augmentation: resize, center crop, color jitter (Brightness, Contrast, Saturation: 0.2), normalize	Nvidia GeForce RTX 2070 (8GB)	23h
EfficientNet B3	0.887	**0.89**	512, 512	3.00E-04	Adam	40	• Loss: Label Smoothing Cross Entropy (0.2) • Train augmentation: random resize and crop, horizontal flip, color jitter (Brightness, Contrast, Saturation: 0.25) • Test augmentation: resize, center crop, color jitter (Brightness, Contrast, Saturation: 0.2), normalize	Nvidia GeForce RTX 2070 (8GB)	23h
EfficientNet B2	0.879	**0.871**	512, 512	5.00E-04	SGD	17	• Loss: Label Smoothing Cross-Entropy (0.1) • Train augmentation: random resize and crop, horizontal flip • Test augmentation: resize, center crop, normalize	Nvidia GeForce RTX 2070 (8GB)	10h
EfficientNet B1	0.889	**0.88**	512, 512	7.00E-04	Adam	14	• Loss: Label Smoothing Cross Entropy (0.2) • Train augmentation: random resize and crop, horizontal flip, color jitter (Brightness, Contrast, Saturation: 0.25) • Test augmentation: resize, center crop, color jitter (Brightness, Contrast, Saturation: 0.2), normalize	Nvidia GeForce RTX 2070 (8GB)	9h
EfficientNet B1	0.881	**0.878**	512, 512	1.00E-03	SGD	14	• Loss: Label Smoothing Cross-Entropy (0.1) • Train augmentation: random resize and crop, horizontal flip • Test augmentation: resize, center crop, normalize	Nvidia GeForce RTX 2070 (8GB)	9h
EfficientNet B1	0.78	**0.757**	240, 240	5.00E-03	Adam	12	• Loss: Label Smoothing Cross-Entropy (0.1) • Train augmentation: random resize and crop, horizontal flip • Test augmentation: resize, center crop, normalize	Nvidia GeForce RTX 2070 (8GB)	8h
EfficientNet B0	0.8862	**0.886**	512, 512	1.00E-03	Adam	15	• Loss: Sparse Cross-Entropy • Callbacks: EarlyStopping (monitored the value of val_loss with a patience of 5 epochs), ReduceLROnPlateau (monitored the value of val_loss with a patience of 2 epochs and a reduction factor of 0.3) • Extra augmentation: rotation (45 degree), zoom (0.8, 1.2), horizontal and vertical flip, shear (0.1 degree), height and width shifting (0.1)	NVidia K80 GPU	6h
EfficientNet B0	0.892	**0.885**	512, 512	1.00E-03	Adam	8	• K-Fold Model 3 • Loss: Sparse Cross-Entropy • Callbacks: EarlyStopping (monitored the value of val_loss with a patience of 5 epochs), ReduceLROnPlateau (monitored the value of val_loss with a patience of 2 epochs and a reduction factor of 0.3) • Extra augmentation: rotation (45 degree), zoom (0.8, 1.2), horizontal and vertical flip, shear (0.1 degree), height and width shifting (0.1)	HPC Kotys	40h
EfficientNet B0	0.875	**0.884**	512, 512	1.00E-0.3	Adam	8	• Loss: Sparse Cross-Entropy • Callbacks: EarlyStopping (monitored the value of val_loss with a patience of 5 epochs), ReduceLROnPlateau (monitored the value of val_loss with a patience of 2 epochs and a reduction factor of 0.3) • Extra augmentation: rotation (45 degree), zoom (0.8, 1.2), horizontal and vertical flip, shear (0.1 degree), height and width shifting (0.1)	HPC Kotys	8h

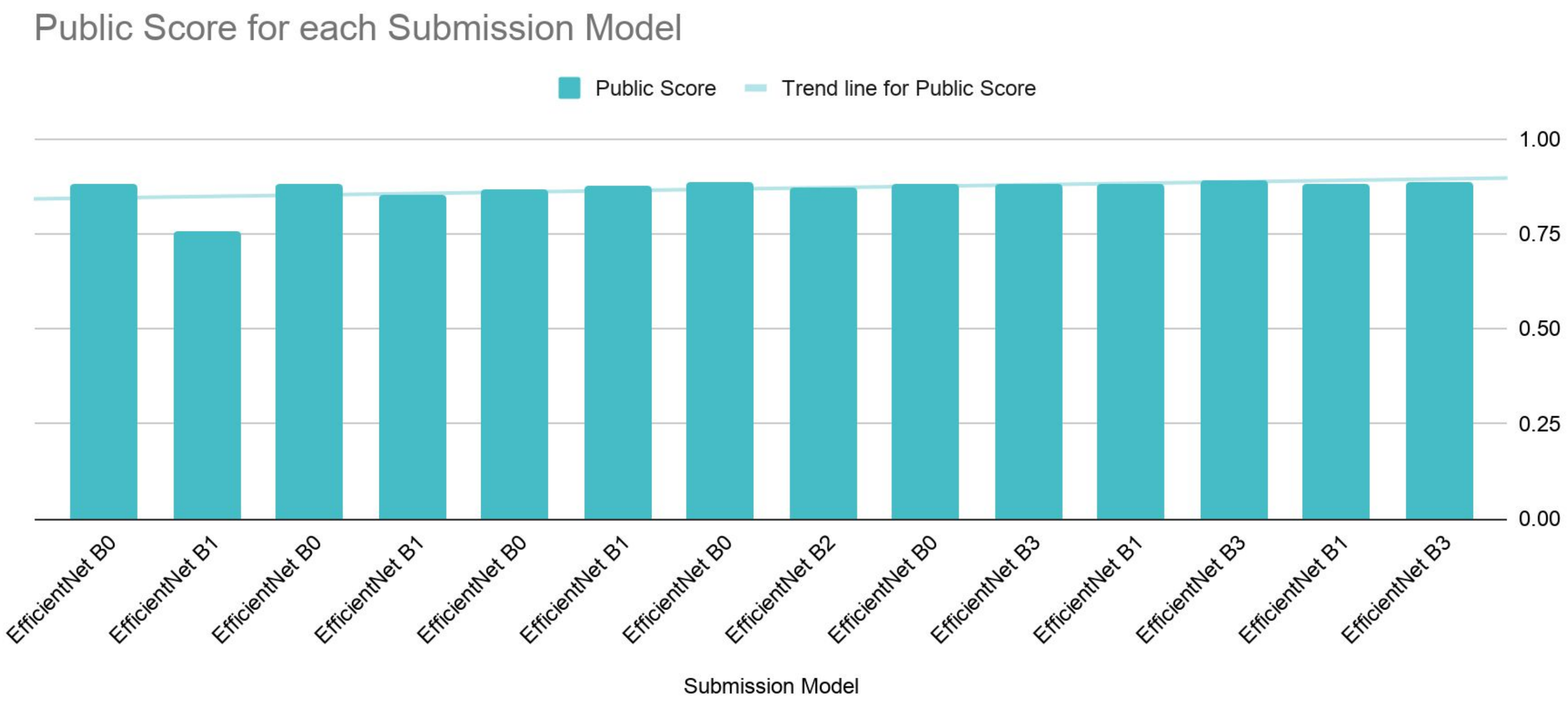
Architecture	Validation accuracy	Public test set score	Input size	Learning rate	Optimizer	Epochs	Notes	Training resources	Time
EfficientNet B0	0.8862	**0.886**	512, 512	1.00E-03	Adam	15	• Loss: Sparse Cross-Entropy • Callbacks: EarlyStopping (monitored the value of val_loss with a patience of 5 epochs), ReduceLROnPlateau (monitored the value of val_loss with a patience of 2 epochs and a reduction factor of 0.3) • Extra augmentation: rotation (45 degree), zoom (0.8, 1.2), horizontal and vertical flip, shear (0.1 degree), height and width shifting (0.1)	NVidia K80 GPU	6h
EfficientNet B3	0.887	**0.89**	512, 512	3.00E-04	Adam	40	• Test Time Augmentation: 5 steps • Loss: Label Smoothing Cross-Entropy (0.2) • Extra augmentation: ColorJitter (Brightness, Contrast, Saturation: 0.25)	Nvidia GeForce RTX 2070 (8GB)	23h
EfficientNet B3	0.887	**0.89**	512, 512	3.00E-04	Adam	40	• Loss: Label Smoothing Cross Entropy (0.2) • Extra augmentation: ColorJitter (Brightness, Contrast, Saturation: 0.25)	Nvidia GeForce RTX 2070 (8GB)	23h
EfficientNet B1	0.889	**0.88**	512, 512	7.00E-04	Adam	14	• Loss: Label Smoothing Cross Entropy (0.2) • Extra augmentation: ColorJitter (Brightness, Contrast, Saturation: 0.25)	Nvidia GeForce RTX 2070 (8GB)	9h
EfficientNet B2	0.879	**0.871**	512, 512	5.00E-04	SGD	17	• Test augmentation: resize, center crop, normalize	Nvidia GeForce RTX 2070 (8GB)	10h
EfficientNet B0	0.892	**0.885**	512, 512	1.00E-03	Adam	8	• K-Fold Model 3 • Loss: Sparse Cross-Entropy • Callbacks: EarlyStopping (monitored the value of val_loss with a patience of 5 epochs), ReduceLROnPlateau (monitored the value of val_loss with a patience of 2 epochs and a reduction factor of 0.3) • Extra augmentation: rotation (45 degree), zoom (0.8, 1.2), horizontal and vertical flip, shear (0.1 degree), height and width shifting (0.1)	HPC Kotys	40h
EfficientNet B1	0.881	**0.878**	512, 512	1.00E-03	Adam	14	• Test augmentation: resize, center crop, normalize	Nvidia GeForce RTX 2070 (8GB)	9h
EfficientNet B1	0.869	**0.853**	512, 512	5.00E-03	SGD	6	• Test augmentation: resize, center crop, normalize	Nvidia GeForce RTX 2070 (8GB)	6h
EfficientNet B1	0.78	**0.757**	240, 240	5.00E-03	Adam	12	• Test augmentation: resize, center crop, normalize	Nvidia GeForce RTX 2070 (8GB)	8h
EfficientNet B0	0.875	**0.884**	512, 512	1.00E-0.3	Adam	8	• Loss: Sparse Cross-Entropy • Callbacks: EarlyStopping (monitored the value of val_loss with a patience of 5 epochs), ReduceLROnPlateau (monitored the value of val_loss with a patience of 2 epochs and a reduction factor of 0.3) • Extra augmentation: rotation (45 degree), zoom (0.8, 1.2), horizontal and vertical flip, shear (0.1 degree), height and width shifting (0.1)	HPC Kotys	8h

Results

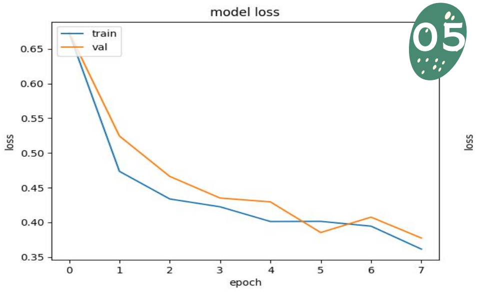
timeline

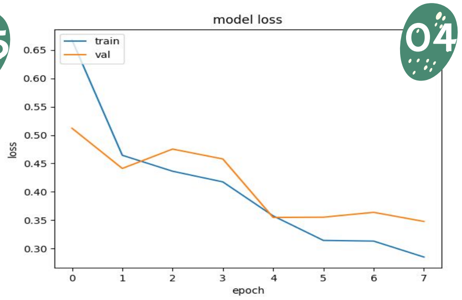
8 epochs

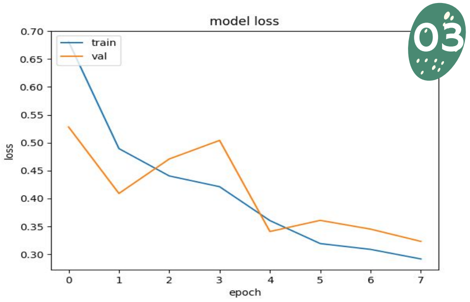
Models		Notes
Efficient Net B0 (from-the-shelf)	**88.4%**	trained a HPC node with 2x Intel Xeon E5-2660 v3 CPU, 10 cores per CPU, 128 GB RAM \ 10 to 40 hours \ batch size: 16, learning rate: xx, [...]
Efficient Net B0 (best **k-fold**)	**88.5%**	
Efficient Net B2	**85.3%**	trained on [...] GPU \ 10 to 40 hours \ batch size: 16, learning rate: xx, [...]
Efficient Net B1	**88.0%**	
Efficient Net B3	**88.2%**	
Efficient Net B3	**89.0%**	

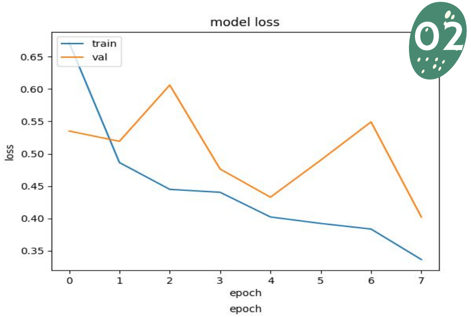

Public Score for each Submission Model
Public Score
Trend line for Public Score
1.00
0.75
0.50
0.25
0.00
EfficientNet B0
EfficientNet B1
EfficientNet B0
EfficientNet B1
EfficientNet B0
EfficientNet B1
EfficientNet B0
EfficientNet B2
EfficientNet B0
EfficientNet B3
EfficientNet B1
EfficientNet B3
EfficientNet B1
EfficientNet B3
Submission Model

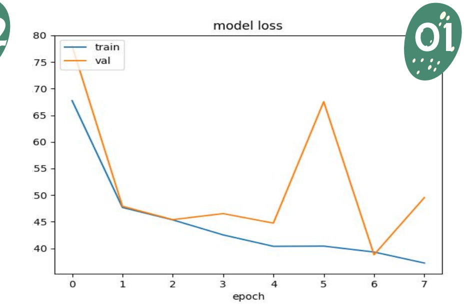
K(5)-Fold Efficient Net loss





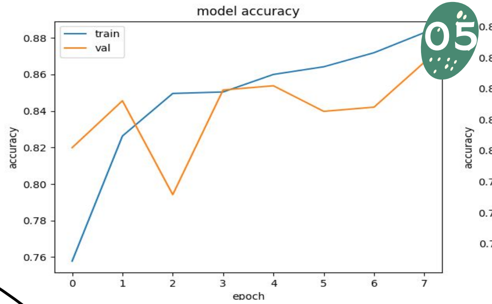


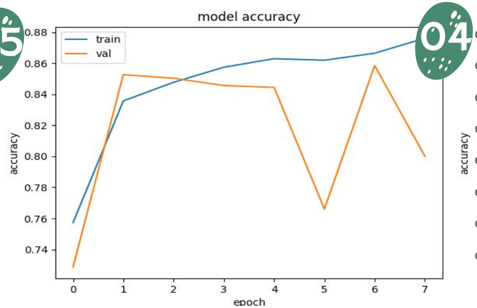


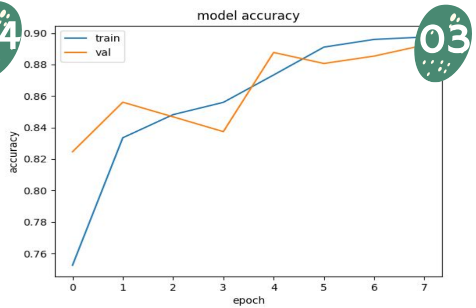


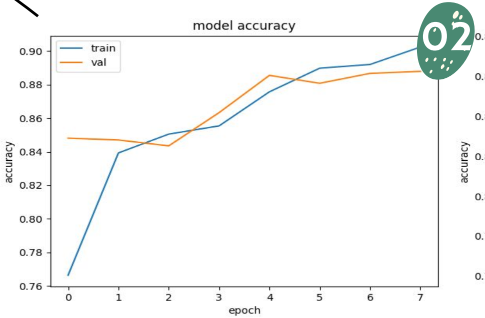
For experimental purposes, we implemented k-fold cross-validation to see the behavior of the model on unknown data. We split our data into 5 batches and trained 8 epochs for every combination of 4 batches for training and one for validation. The training was made on the HPC Kotys, and took 40 hours.

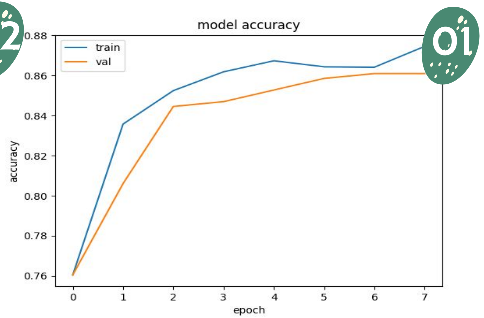
K(5)-Fold Efficient Net accuracy







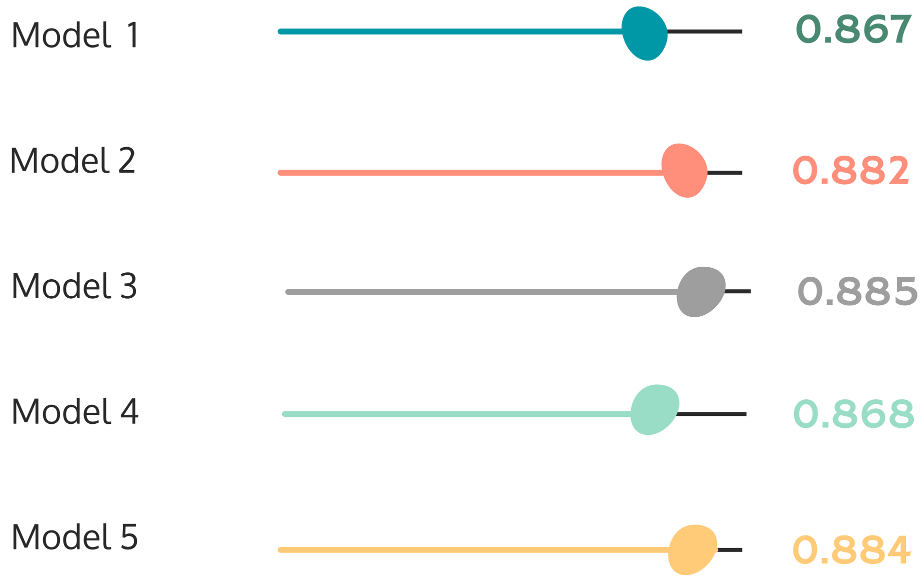




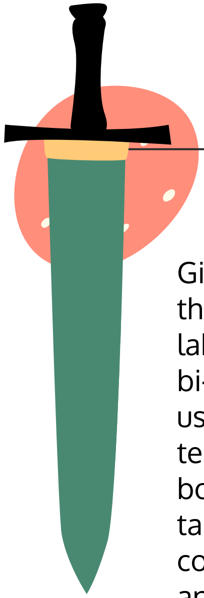
The results after k-fold cross-validation show that the first epochs of training have spikes and higher accuracy for validation than for training. The behavior did not indicate a bad thing, but a curios effect of either dropouts or augmentation present on training and not on testing.

We computed the confidence interval on accuracy and our conclusion is that "We are 95% confident that the value of accuracy after 8 epochs is between 0.867 and 0.878".

K(5)-Fold Efficient Net submissions



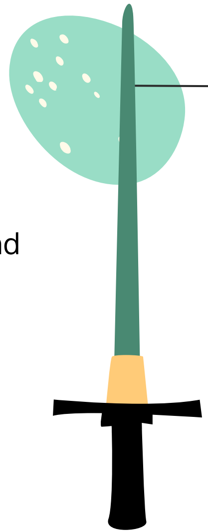
Further improvements and paths



#1

Bi-tempered loss

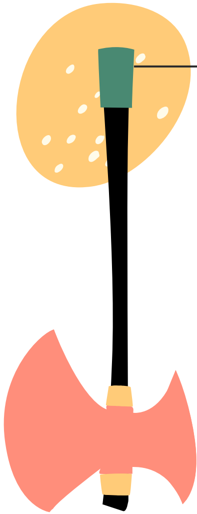
Given the discussions around the correctness of the data labels, we can use the bi-tempered loss. This loss uses 2 parameters called temperatures to control boundedness and tail-heaviness and, thus, compensate for both small and large margin noise.



#2

2x Classifier for data imbalancement

Given the imbalance of the training set, one option is to create 2 classifiers: a binary classifier between the dominant class and all the other, followed by a classifier for all the non-dominant classes.



#3

Finding better training resources

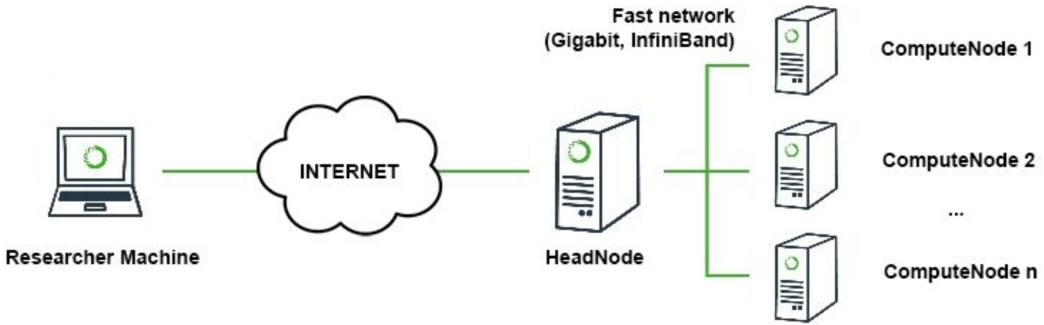
Transitioning from CPU to GPU on the HPC nodes, so that we may allow many more epochs to take place.

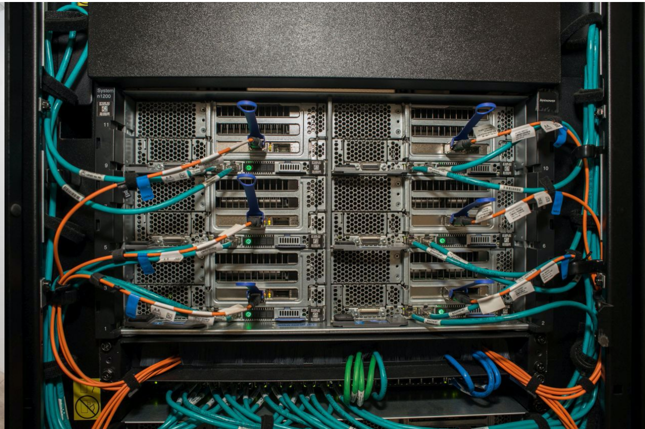
Thank you for
your attention!

Additional materials.

University's resources for machine learning

The Babeș-Bolyai University offers access to its High Performance Center, offering 68 nodes with high end 10 cores processors, K40X GPUs and dedicated Intel Phi coprocessors. In this project all actions taken on Kotys were managed manually by the team members, following the university guideline and security protocols.





Training on Kotys IBM Machine

data transfer

Terminal 1: **ssh** -L local_port:kotys.cs.ubbcluj.ro:remote_port stud_credentials@www.scs.ubbcluj.ro -p stud_server_port
(running in background)

Terminal 2: **sftp** -oPort=local_port stud_credentials@127.0.0.1
(initiate data transfer)

Secure file transfer to the bigdata partition

Port tunneling into the university network, then towards hpc

CPU usage - brief setup

ssh stud_credentials@www.scs.ubbcluj.ro -p stud_server_port
[stud_credentials@linux ~]$ **ssh** -p hpc_port kotys.cs.ubbcluj.ro
[stud_credentials@kotys ~]$ **ssh** compute056
(the user must check whether any process is already running at the time they are accessing the node)

[stud_credentials@compute056 ~]$ **cd** /bigdata/users-data/stud_folder
(anaconda must be installed before performing the following steps)

[stud_credentials@compute056 stud_folder]$ export PATH=/anaconda3/bin:$PATH
[stud_credentials@compute056 stud_folder]$ conda create -n cassava_env --clone=/bigdata/users-data/stud_folder/anaconda3
[stud_credentials@compute056 stud_folder]$ conda activate cassava_env
(cassava_env) [stud_credentials@compute056 stud_folder]$ conda install tensorflow
(cassava_env) [stud_credentials@compute056 stud_folder]$ python efficientnet.py