# **Cassava Leaf Disease Classification Report**

My work has been splitted into several consecutive stages. It will be easier to understand what is happening at a certain stage if you look at the previous ones. All the successful and not ideas are listed at the end. At the beginning, I will use a small model for faster experiments and replace it with a better model later.

**Throughout the competition, when classifying approaches, I will look mainly at validation metrics and almost ignore the public LB score.**

**Hardware:** RTX 3090 x 1 (BS x 4), RTX 3080 x 2 (BS x 1), Google Colab Pro x 3 (BS x 2)

**I. Initializing default pipeline**

* Model - ResNet18
* Pretrained - True
* Image size - 256
* Epochs - 12
* Scheduler - None
* BatchSize - 128
* Augmentations - Horizontal Flip (p=0.5)
* Apex - turned off
* Train, val, test split - (0.9, 0.1, 0.0)
* Optimizer - AdamW (lr = 0.001)
* Loss Function - CrossEntropyLoss
* Metric - F1-score (weighted)

**II. Optimizer search report**

**Metrics value was taken from the epoch with the lowest validation loss.**

**1st round (scheduler - None, epoch - 12)**

**All the default optimizers**

|  |  |  |  |
| --- | --- | --- | --- |
| **Optimizer** | **Learning rate** | **Validation Loss** | **Metrics** |
| SGD | 0.01 | 0.5400 | 0.8211 |
| Adadelta | 1.0 | 0.5947 | 0.7946 |
| Adagrad | 0.01 | 0.5293 | 0.8087 |
| Adam | 0.001 | 0.5239 | 0.8285 |
| AdamW | 0.001 | 0.5396 | 0.8044 |
| Adamax | 0.02 | 0.5280 | 0.8266 |
| ASGD | 0.01 | 0.5417 | 0.8111 |
| RMSprop | 0.01 | 0.7804 | 0.6859 |
| Rprop | 0.01 | 0.9642 | 0.5285 |

**2nd round (scheduler - Cosine Decay, epoch - 12)**

**Adding improved optimizers, excluding the bad ones, some default learning rates are changed.**

**Cosine Decay: 1/2 \* (1 + cos(STEP \* math.pi / STEPS)) \* lr**

**The number of steps was calculated for 120 epochs, otherwise the learning rate would have fallen very quickly**

|  |  |  |  |
| --- | --- | --- | --- |
| **Optimizer** | **Learning rate** | **Validation Loss** | **Metrics** |
| Adamax | 0.002 | 0.4873 | 0.8447 |
| SGD | 0.02 | 0.5277 | 0.8162 |
| ASGD | 0.02 | 0.5448 | 0.8088 |
| SGDP (Nesterov=True) | 0.02 | 0.5140 | 0.8140 |
| SGDP (Nesterov=False) | 0.02 | 0.5342 | 0.8326 |
| AdamP (Nesterov=True) | 0.001 | 0.5298 | 0.8386 |
| AdamP (Nesterov=False) | 0.001 | 0.5040 | 0.8259 |
| Adam | 0.002 | 0.4445 | 0.8458 |
| AdamP (Nesterov=False) | 0.002 | 0.4820 | 0.8399 |

**3rd round (scheduler - Cosine Decay, epoch - 12)**

**Rerunning the best ones.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Optimizer** | **Learning rate** | **Validation Loss** | **Metrics** | **LB score** |
| Adam | 0.002 | 0.4748 | 0.8359 | 0.842 |
| Adamax | 0.002 | 0.5021 | 0.826 | 0.826 |
| SGDP (Nesterov=True) | 0.02 | 0.5097 | 0.8289 | 0.830 |
| AdamP  (Nesterov=False) | 0.002 | 0.4721 | 0.8469 | 0.849 |

**4rd round (scheduler - Cosine Decay, epoch - 12)**

**One more round for the two best optimizers. I tried to choose the best one trying each of them with the different learning rates.**

**Before that, I used the validation loss as a target metric, I used it for an early stopping and focused on it to see the potential of the optimizers. From now on, I will always use F1-score as the target metric.**

**Validation loss value was taken from the epoch with the highest F1-score.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Learning rate** | **Adam metrics** | **AdamP metrics** | **Adam**  **Val loss** | **AdamP**  **Val loss** |
| 0.1 | 0.7070 | 0.7349 | 0.7222 | 0.6749 |
| 0.01 | 0.7585 | 0.7692 | 0.6203 | 0.5997 |
| 0.001 | 0.8285 | 0.8259 | 0.5239 | 0.5040 |
| 0.0001 | 0.8290 | 0.8276 | 0.5999 | 0.6198 |
| 0.00001 | 0.7991 | 0.7912 | 0.5757 | 0.5798 |
| 0.002 | 0.8359 | 0.8469 | 0.4748 | 0.4721 |
| 0.003 | 0.8493 | 0.8332 | 0.4439 | 0.4728 |
| 0.005 | 0.7963 | 0.7881 | 0.5374 | 0.5643 |
| 0.0005 | 0.8571 | 0.8527 | 0.5936 | 0.5937 |

Now we can definitely say that tuned Adam is better than tuned AdamP. The best score for AdamP was achieved with lr=0.002 and equals 0.8469 (valloss=0.4748). The best score for Adam is 0.8571, but it is the overfitted one. We can check it if we look at valloss (0.5936) and LB score (0.838) which usually has almost perfect correlation with validation.The next best score for Adam is 0.8493, it’s not much better than AdamP’s best score, but it has much better validation loss (0.4439 vs 0.4748). Then I gave AdamP one more chance and trained it with nesterov=True and the best learning rate. Unfortunately, it hasn’t improved the score (metrics=0.8328, valloss=0.4675).

**Round 5 (scheduler - Cosine Decay, epoch - 12)**

**Final round for tuning the best optimizer.**

1. I tried Adam with 0.0003 lr, but it overfitted fast. lr=0.004 didn’t show good results as well.
2. I tried Adam with different weights\_decays (default=0), lr=0.003

|  |  |  |
| --- | --- | --- |
| **Weight Decay** | **Metrics** | **Validation Loss** |
| 0.01 | 0.7382 | 0.6649 |
| 0.03 | 0.6890 | 0.7505 |
| 0.001 | 0.8278 | 0.4869 |
| 0.005 | 0.7630 | 0.6085 |
| 0.0001 | 0.8248 | 0.4897 |

Changing weight decay hasn’t improved the score.

1. Trying Adam (lr=0.003) with amsgrad=True.

Metrics: 0.8239

Validation loss: 0.4780

**Optimizer Search Results:**

**Source:**

* Optimizer - AdamW
* Learning rate - 0.001
* Metrics - 0.8044
* Validation loss - 0.5396

**Final:**

* Optimizer - Adam
* Learning rate - 0.003
* Metrics - 0.8493
* Validation loss - 0.4439

I trained the model a few times with the different seeds and the metrics error was about 0.01.

**III. Training set up & tuning**

In this section, I will try to finally set up the training process: choose the optimizer, loss function, scheduler, model, starting augmentations and other training parameters. In the next section, I will start working with data and augmentations.

1. Since this moment I will always use at least 21 epochs to train models with early\_stopping=6.
2. I tried MobileNetV2 and it had better performance than resnet18 with almost the same training speed (or even faster). So further I will use this model. In one of the next stages I will do a model zoo to compare all the models.
3. I tried a custom loss function with label smoothing (0.05) and it had better performance than simple CrossEntropyLoss. So further I will use this loss function.
4. When I was monitoring learning rate changes I noticed that it drops too fast, so I decided to turn on the scheduler after a different number of epochs. Here are a little report (using MobileNetV2):

|  |  |  |
| --- | --- | --- |
| **Start epoch** | **Metrics** | **Validation Loss** |
| 1 | 0.8628 | 0.5945 |
| 4 | 0.8644 | 0.4212 |
| 5 | 0.8633 | 0.4139 |
| 6 | 0.8574 | 0.4516 |

The best performance was achieved with start\_epoch=4.

1. I tried label smoothing with greater smoothing value (0.05 >> 0.1). It has increased validation and LB scores (0.8644 >> 0.8677, 0.865 >> 0.869). However, loss has increased (0.4212 >> 0.7428), but that’s fine because we can assume that we use a different one, so the loss standard has been changed after this point.
2. I also noticed that when the learning rate is calculated with the formula above, it also drops very fast with bigger models, because it depends on the number of steps, but this value changes with changing batchsize. So I decided to write to my own scheduler which takes into account batchsize. As parameters, I also added a minimum learning rate (I set 1e-8) and start\_epoch (I set 4).
3. I decided to use Vertical Flip Augmentation. Metrics increased: 0.8677 >> 0.8699
4. I decided to increase smoothing value in the loss function

twice:

The first time (0.1 >> 0.15): 0.8699 >> 0.8731

The second time (0.15 >> 0.2): 0.8731 >> 0.8741

The 0.2 smoothing value has shown the best performance, but

it has set a new validation loss value - 0.9594.

1. I tried Ranger and Diff Grad optimizers with different learning rates. The best performance was achieved with Ranger optimizer (lr=0.003). Metrics: 0.8741 >> 0.8775.
2. Then I applied Vertical Flip augmentation. I put it in this section, because this augmentation multiplies the amount of unique images and needs more time for training so I had to tune my scheduler again - now the standardbs parameter equals 256 instead of 128. Metrics: 0.8775 >> 0.8849.
3. I tried the scheduler from the following notebook: [Baseline - Modified From Previous Competition](https://www.kaggle.com/haqishen/baseline-modified-from-previous-competition), but it has shown worse performance. Except that this scheduler has a warm-up, the difference is that in it a step is made every epoch, and not every batch as I have, so the learning rate decreases more slowly.
4. I’ve taken my own scheduler again and decided to apply warm up to it. Metrics: 0.8849 >> 0.8874.

**Training set up & tuning Results:**

**Source:**

* Epochs - 12
* Model - ResNet18
* Optimizer - Adam (lr=0.003)
* Loss function - CrossEntropyLoss
* Scheduler - CosineDecay
* Validation loss - 0.4439
* Metrics - 0.8493
* LB score - 0.849

**Final:**

* Epochs - 21
* Model - MobileNetV2
* Optimizer - Ranger (lr=0.003)
* Loss function - custom CrossEntropyLoss with Label Smoothing (smoothing=0.2)
* Scheduler - Cosine Batch Decay Scheduler (custom) + Warm Up
* Validation loss - 0.9537 (new loss function)
* Metrics - 0.8874
* LB score - 0.871

**IV. The best resize search**

There are a lot of different ways to resize images, it might be resizing itself or different crops. Here I'm gonna try to find the best one. Number of epochs increased to 30, early stopping is 7 epochs.

1. Simple Resize 256 \* 256 - default on this moment.
2. Random crop with final image size 256 \* 256. It hasn’t shown any good results. With resizes to 320, 352, 384 before cropping the image, results are worse. With resize to 288 the score is the same. Conclusion: useless here.

3-7

|  |  |  |  |
| --- | --- | --- | --- |
| **Approach** | **Metrics** | **Val Loss** | **LB** |
| Simple resize 512 \* 512 | 0.8874 >> 0.8909 | 0.9329 | 0.871 >> 0.883 |
| 512 \* 512 random crop | 0.8678 | 0.9921 | - |
| 512 \* 512 random crop and then resize to 256 \* 256 | 0.8640 | 0.9666 | - |
| 512 \* 512 center crop | 0.8806 | 0.9522 | - |
| 512 \* 512 center crop and then resize to 256 \* 256 | 0.8665 | 0.9733 | - |

8. Progressive image size. 3 epochs to warm up with 256 size, then image size starts increasing every epoch on 32, after 8 epochs it reaches 512 size and scheduler turns on. Metrics: 0.8909 >> 0.8946. LB: 0.883 >> 0.884. Val Loss: 0.9329 >> 0.9315

9. Progressive image size. 3 epochs to warm up with 256 size, then image size starts increasing every epoch on 32, after 8 epochs it reaches 512 size but scheduler turns on already after 3 epochs counting from the end of the warm-up. Thus, the scheduler turns on even before the image reaches its maximum size. Metrics: 0.8922 (worse).

10. Progressive image size. 3 epochs to warm up with 256 size, then image size starts increasing every epoch on 64, after 4 epochs it reaches 512 size and scheduler turns on. Metrics: 0.8935 (worse).

11. Progressive image size. 3 epochs to warm up with 256 size, then image size starts increasing every epoch on 128, after 2 epochs it reaches 512 size and after 3 epochs the scheduler turns on. Metrics: 0.8840 (worse).

**The best resize search Results:**

**Source:**

* Epochs - 21
* Model - MobileNetV2
* Optimizer - Ranger (lr=0.003)
* Loss function - custom CrossEntropyLoss with Label Smoothing (smoothing=0.2)
* Scheduler - Cosine Batch Decay Scheduler (custom) + Warm Up
* Image size - 512
* Validation loss - 0.9537
* Metrics - 0.8874
* LB score - 0.871

**Final:**

* Epochs - 30
* Model - MobileNetV2
* Optimizer - Ranger (lr=0.003)
* Loss function - custom CrossEntropyLoss with Label Smoothing (smoothing=0.2)
* Scheduler - Cosine Batch Decay Scheduler (custom) + Warm Up
* Image size - progressive (256, 288, …, 512)
* Validation loss - 0.9315
* Metrics - 0.8946
* LB score - 0.884

**V. TTA**

Test time augmentations

1. 4 steps average
2. Default
3. Horizontal Flip
4. Vertical Flip
5. Horizontal + vertical flips

LB: 0.884 >> 0.889

II. 16 steps average

0, 90, 180 and 270 degrees rotates for each other.

LB: 0.888 (worse)

III. 4 steps weighted. 0.3, 0.3, 0.2, 0.2 for 1-4

Accordingly. LB: 0.888 (worse)

IV. 16 steps weighted. 1 / 8 for images without rotates and 1 / 24 for images with rotates. LB: 0.889 >> 0.891.

**TTA Results:**

**LB:** 0.884 >> 0.890 (with 16 weighted steps)

**VI. Augmentations + Loss Function search**

In this section I’ll try different augmentations, transforms and other data tricks.

By this point I am using:

* Progressive image size
* Normalize
* Horizontal Flip (p=0.5)
* Vertical Flip (p=0.5)
* Random Rotate (p=1, it might not rotate)

Current best metrics: 0.8946

|  |  |  |  |
| --- | --- | --- | --- |
| **Approach** | **Parameters** | **Metrics** | **ValLoss** |
| Random  Brightness  Contrast | brightness\_limit=0.1,  contrast\_limit=0.15,  p=0.5, | 0.8874 | 0.9332 |
| ShiftScale  Rotate | shift\_limit=0.1, scale\_limit=0.1, rotate\_limit=0, p=0.5, | 0.8936 | 0.9335 |
| HueSaturation  Value | hue\_shift\_limit=20, sat\_shift\_limit=30, val\_shift\_limit=20, p=0.5, | 0.8940 | 0.9338 |
| CoarseDropout | max\_holes=8, max\_height=int(14 \* imgsize / 512),  max\_width=int(14 \* imgsize / 512),  p=0.5 | 0.8810 | 0.9426 |
| Fmix | Alpha=3.0, decay\_power=1.0 | 0.8942 | 0.9341 |
| CutMix | alpha=1.0 | 0.8894 | 0.9456 |
| MixUp | alpha=1.0 | 0.8927 | 0.9379 |

Neither approach improved the score. Most likely, the fact is that the training goes too few epochs. Next, we will try to extend the training by increasing the number of epochs and slowing down the scheduler. We will do it for default MobileNet (without additional augmentations) for comparison.

MinLr: 1e-7 >> 1e-8

Early stopping: 7 >> 10

By augs I mean all 7 above except CoarseDropout (it’s shown the worst performance).

|  |  |  |
| --- | --- | --- |
| **Approach** | **Metrics** | **VaLLoss** |
| 40 epochs | 0.8868 (strange) | 0.9386 |
| 40 epochs + StandardBS 256 >> 512 | 0.8909 | 0.9355 |
| 40 epochs + StandardBS 256 >> 512 + only horizontal flip on warm-up | 0.8729 | 0.9684 |
| 40 epochs + augs | 0.8933 | 0.9401 |
| 40 epochs + augs + StandardBS 256 >> 512 | 0.8898 | 0.9371 |
| 40 epochs + StandardBS 256 >> 512 + augs + only horizontal flip on warm-up | 0.8990 | 0.9373 |

I’ve submitted that last one and got 0.892 on LB which is an improvement of the score.

Then I rerun this training with standardbs=256, but I got not as good results - 0.8950 only. I’ve also tried to apply 5 epoch warm-up above this, but the score was worse again - 0.8915.

During the break I applied the mish-activation instead of ReLu, cause the first one is recommended as a good combination with Ranger optimizer. However, the results were much worse, metrics achieved only 0.8650 value.

Then I tried BiTempered Loss with the different t1 and t1 values for different types of noise.

Smoothing=0.2 everywhere.

|  |  |
| --- | --- |
| **t1, t2** | **Metrics** |
| 0.2, 4.0 | 0.8667 |
| 1.0, 4.0 | 0.8695 |
| 0.2, 1.0 | 0.8928 |

It tried the best one with the different smoothing values but it hasn’t set a new record.

Then it remains to test Symmetric Cross Entropy Loss. I did with both types of reduction: sum and mean. The first one got 0.8918 metrics on validation, the second one got 0.8953 metrics on validation and 0.885 on LB.

**Merge + Loss Function search results:**

**Source:**

* Epochs - 21
* Model - MobileNetV2
* Optimizer - Ranger (lr=0.003)
* Loss function - custom CrossEntropyLoss with Label Smoothing (smoothing=0.2)
* Scheduler - Cosine Batch Decay Scheduler (custom) + Warm Up
* Image size - 512
* Validation loss - 0.9315
* Metrics - 0.8946
* LB score - 0.890

**Final:**

* Epochs - 40
* Model - MobileNetV2
* Optimizer - Ranger (lr=0.003)
* Loss function - custom CrossEntropyLoss with Label Smoothing (smoothing=0.2)
* Validation loss - 0.9373
* Metrics - 0.8990
* LB score - 0.892
* **New features:**
  + MinLr: 1e-7 >> 1e-8
  + Early stopping: 7 >> 10
  + New augmentations have been added
  + Only horizontal flip augmentation is applied during the learning rate warm-up
  + StandardBS decay 256 >> 512

**VII. 2019 data usage**

I tried using 2019 data. I decided that the best way to do it is to use it only in a train dataset and avoid putting it into a validation one. At the same time I tried to use it as a tool for dealing with the class imbalance, so I trained it once with all the images from there (except for duplicates) and once with all the images except the most frequent class.

|  |  |  |  |
| --- | --- | --- | --- |
| **Approach** | **Metrics** | **ValLoss** | **LB score** |
| Full 2019 data + 512 StandardBS | 0.8944 | 0.9319 | 0.891 |
| 2019 data except for the most frequent class + 512 standardBS | 0.8971 | 0.9271 | 0.887 |
| Full 2019 data + 1024 StandardBS | 0.8986 | 0.9268 | 0.890 |
| 2019 data except for the most frequent class + 1024 standardBS | 0.8949 | 0.9311 | 0.886 |

All the methods have shown almost the same validation metrics as before, but losses are lower. LB score is strange again so I won’t evaluate these training on it, keeping trust in my CV. Next I will use the “Full 2019 data + 1024 StandardBS” approach.

**VII. A new bunch of methods and tricks**

1. I changed the Label Smoothing Loss class so the smoothing was applied only in 50% cases. Metrics: 0.8948 (worse).
2. In the previous stage I turned off heavy augmentations and applied it again. Metrics: 0.8970. LB: 0.889. It’s a little worse (LB is worse as well). But I will keep using it because it has shown a good performance before. At the same time I understand that my current model might be too small to work with such heavy augmentations so there is a chance that they will show better performance later with the bigger models.
3. I tried progressive batchsize starting from 2 and ending on 32. Metrics: 0.8845 (worse).
4. Focal Loss (alpha=0.25, gamma=2). Metrics: 0.8916 (worse).
5. Frozen BN layers. Metrics: 0.8986. LB: 0.889. I’ll keep using it next.
6. I tried to use max possible BS on each epoch. It makes sense since I’m using progressive image size. Metrics: 0.8958 (worse).
7. I noticed that the learning rate decreases too fast so I reduced the StandardBS parameter for scheduler from 1024 to 512. Metrics: 0.8970 >> 0.8999
8. Blur (p=0.5, blur\_limit=2.5). Metrics: 0.8999 >> 0.9020. Val Loss: 0.9367. LB: 0.885
9. SAM optimizer above the Ranger. Metrics: 0.8938 (worse).
10. ElasticTransform (alpha=1,sigma=50,alpha\_affine=50,interpolation=1,border\_mode=4,p=0.5). Metrics: 0.8951 (worse). And 4x slower.
11. GridDistortion

(num\_steps=5, distort\_limit=0.3,interpolation=1,border\_mode=4, p=0.5).

Metrics: 0.8953 (worse).

1. Optical distortion. Metrics: 0.9005 (worse).
2. Turning off the blue channel in image. Metrics: 0.9002 (worse).
3. RandomSunFlare. Interrupted after 11 epoch.
4. GaussNoise. Metrics: 0.8955 (worse).
5. I tried 360 rotates instead of discrete 90 rotates combining it with different shift scale rotates and probabilities. Max score on validation was 0.8950 which is lower than it was before. Apparently, such a strange result was caused by using a too small model. I’ll come back to these methods later with the bigger models.

**Source:**

* Validation loss - 0.9268
* Metrics - 0.8986
* LB score - 0.890

**Final:**

* Validation loss - 0.9367
* Metrics - 0.9020
* LB score - 0.885

**VIII. Using Apex and preparing for training a Zoo**

Epochs - 30

* Without Apex - 0.8833
* Mixed Precision - 0.8934
* Mixed Precision + Gradient Accumulation (iters\_to\_accumulate = 2) - 0.8978
* Mixed Precision + Gradient Accumulation (iters\_to\_accumulate = 4) - 0.9004
* Mixed Precision + Gradient Accumulation (iters\_to\_accumulate = 8) - 0.9022, LB = 0.890

I retrained the last one with 50 epochs and got worse results - 0.8978 metrics in validation, so since this moment I will use 30 epochs for all models.

**Source:**

* Validation loss - 0.9367
* Metrics - 0.9020
* LB score - 0.885

**Final:**

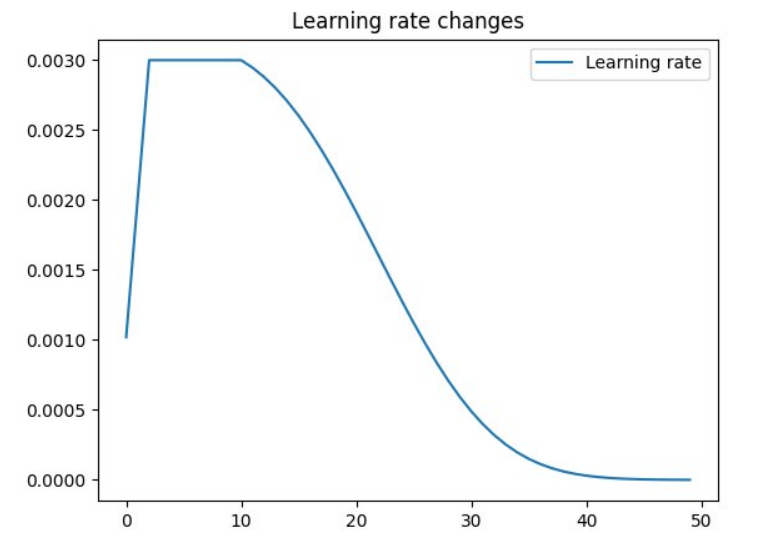
* Validation loss - 0.9285
* Metrics - 0.9022
* LB score - 0.890

**IX. Model Zoo report**

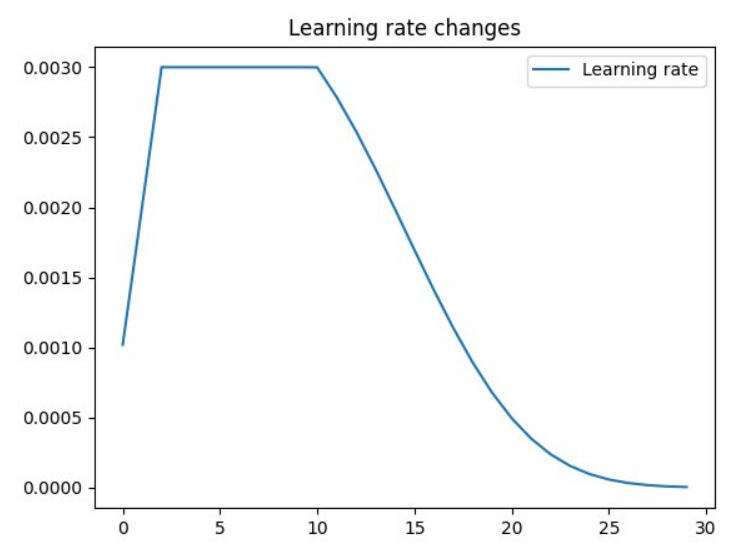
**Initializing current pipeline:**

* Pretrained - True
* Image size - progressive (start = 256, final = 512, step = 32)
* Epochs - 30
* Scheduler - Cosine Batch Decay Scheduler (custom) + Warm Up. Parameters:
  + epochs=120,
  + standardbs=64,
  + startepoch=9,
  + minlr=1e-8,
* Augmentations:
  + 1-3 (warm up) epochs - Horizontal Flip (p=0.5)
  + 3 - final:
    - HorizontalFlip(p=0.5)
    - VerticalFlip(=0.5)
    - RandomBrightnessContrast (brightness\_limit=0.1, contrast\_limit=0.15, p=0.5)
    - ShiftScaleRotate (shift\_limit=0.1,scale\_limit=0.1,rotate\_limit=0,p=0.5)
    - HueSaturationValue (hue\_shift\_limit=20, sat\_shift\_limit=30, val\_shift\_limit=20, p=0.5)
    - Blur (blur\_limit=2.5 p=0.5)
* Apex, mixed precision, gradient accumulation (iters\_to\_accumulate=8)
* Train, val, test split - (0.9, 0.1, 0.0)
* Optimizer - Ranger (lr = 0.003)
* Loss Function - CrossEntropyLoss with Label Smoothing (smoothing = 0.2)
* Metric - F1-score (weighted)
* Frozen BN layers
* 2019 data (used only in the train dataset)
* No TTA
* Max possible batch size

**Previous learning rate changes plot (50 epochs):**

****

**New learning rate changes plot (50 epochs):**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Metrics** | **ValLoss** | **LB score** | **Batchsize** |
| Mobile Net V2 | 0.9022 | 0.9285 | 0.890 | 32 |
| Resnet 50 | 0.9049 | 0.9251 | 0.892 | 16 |
| resnext50\_32x4d | 0.9029 | 0.9255 | 0.895 | 16 |

**New thoughts:**

By that moment I noticed two things.

First, the 4x and 16x aren’t stable and might often decrease score. Moreover, they take too much time that might be used for a bigger ensemble in inference. So I decided to use a simple 2x TTA which averages to predictions: source image and a horizontal flipped one. Looking ahead, I will say that it allowed me to achieve 0.900 accuracy in the public LB with just a single fold.

I also noticed that the learning mode when the scheduler turns on the 9th epoch after warm up isn’t good for bigger models. It worked well only with a small model like a MobileNet. So I set the start epoch for scheduler equaled 4 instead of 9 (and 1 for large models like an efficient net). Other scheduler parameters left the same so the plot above is still actual, except for one parameter.

Next, I will train all models with these parameters and make predictions with the new TTA. I’m also forced to retrain the best ones of ones which have already been tested.

Here is the table with the data about training models with the new parameters.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Params** | **Metrics** | **ValLoss** | **LB score** |
| vit\_base\_patch16\_384 | default | 0.8713 | 0.9533 | - |
| vit\_base\_patch16\_384 | lr=0.00003 (after warm up) | 0.8912 | 0.9547 | 0.887 (No TTA) |
| resnet50 | default | 0.9049 | 0.9251 | 0.892 (No TTA) |
| resnext50\_32x4 | default | 0.9029 | 0.9255 | 0.895 (No TTA) |
| resnext101\_32x8 | default | 0.9055 | 0.9126 | 0.900 |
| resnext101\_32x8 | dropout=0.75 | broken | broken | - |
| resnext101\_32x8 | dropout=0.5 | broken | broken | - |
| efficientnet\_b3 | start\_epoch=1 | 0.8982 | 0.9285 | 0.888 (No TTA) |
| swsl\_resnext101\_32x8d | default | 0.9060 | 0.9119 | 0.893 |
| swsl\_resnext101\_32x16d | default | 0.8992 | 0.9288 | 0.888 (No TTA) |
| swsl\_resnext101\_32x4d | default | 0.9066 | 0.9179 | 0.895 |
| ecaresnet101d\_pruned | default | 0.9015 | 0.9201 | - |
| seresnext50\_32x4d | start\_epoch=1 | 0.8967 | 0.9234 | 0.888 |
| swsl\_resnext50\_32x4d | default | 0.9072 | 0.9238 | 0.895 |
| res2net101\_26w\_4s | default | 0.9000 | 0.9272 | - |
| resnest50d\_1s4x24d | default | 0.8974 | 0.9314 | - |
| vit\_large\_patch32\_384 | lr=0.00003, start\_epoch=1 | broken | broken | - |
| tv\_densenet121 | default | 0.8958 | 0.9559 | - |
| vit\_large\_patch32\_384 | lr=0.0000003  start\_epoch=1  standard bs=32 | 0.8731 | 0.9665 | - |
| swsl\_resnext101\_32x8d | standard bs=32,  dropout=0.25 | 0.9064 | 0.9157 | - |
| inception\_v4 | dropout=0.25 | 0.8974 | 0.9276 | - |
| densenet161 | dropout=0.25 | 0.8942 | 0.9575 |  |
| Efficient Net V4 | default | 0.8967 | 0.9442 | - |
| inception\_v4 | default | 0.8981 | 0.9335 | - |
| vit\_base\_patch16\_384 | lr=0.00003 (after warm up), 360 degrees rotates + coarse dropout | 0.8787 | 0.9547 | - |
| Efficient Net V4 | standard bs=32 | 0.8999 | 0.9236 | - |
| resnext101 | standard bs=32 | 0.9064 | 0.9157 | - |

**Source:**

* Validation loss - 0.9285
* Metrics - 0.9022
* LB score - 0.890

**Final:**

* Validation loss - 0.9126
* Metrics - 0.9072
* LB score - 0.900

**X. K-fold, final tricks, external data and the secret weapon**

In this part I’ll train K-fold based on swsl\_resnext101\_32x8d for the first time, will try a couple of final augmentations and parameters, will try external data and the secret weapon - knowledge distillation.

**1. K-fold**

Just as in the case of simple train-test split 2019 data won’t be used in validation, and will be used only in the train part. I added it manually in each fold after the split. All the parameters are the same.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Fold | I | II | III | IV | V |
| Metrics | 0.8862 | 0.8955 | 0.8973 | 0.8825 | 0.8947 |

Average: 0.89124

LB: 0.896

**2. Knowledge distillation**

Since the dataset is very noisy, relabeling the data was unavoidable.

The algorithm is simple. I took folds one by one and predicted labels for each image from the validation part of this fold. If the label didn’t match the source labeled and confidence was above the threshold, I changed this label to a new one.

How did I choose the confidence threshold? I searched for an approximately number of wrong labels in the data in the discussions and found out that there are about 500 diseased plants labeled as healthy and a small bunch of information about labeling healthy images as diseased. So I’ve decided to process only those cases where diseased plants might be labeled as healthy. I started finding a threshold with which about 500 labels would be changed. So it was 0.55. It’s NOT ranged from 0 to 1, it has much more amplitude including negative values.  
  
The 2019 part of the data was relabeled as well.

After relabeling the dataset with the resnext101 folds I trained Efficient Net V4 with the new labels. Understanding that the number of noisy labels had to be decreased, I remembered to try to decrease label smoothing value.

|  |  |  |  |
| --- | --- | --- | --- |
| Source | KD, Smoothing=0.0 | KD, Smoothing=0.2 | KD, Smoothing=0.1 |
| 0.8999 | 0.9082 | 0.9013 | 0.9126 |

**3. External data**

Talking about external data I mean [Mendeley Leaves](https://www.kaggle.com/nroman/mendeley-leaves). This dataset contains images of other plants. The dataset is not fully labeled. The first part of the all images are labeled just as healthy and the rest of them are labeled just as diseased. So the second part had to be pseudo-labeled to be able to use it for training. I used 5 folds described above for this task. I predicted labels for each diseased image and checked model confidence for each one. I used a 0.55 confidence threshold (the reason for taking this threshold is described above). Finally, I get 2767 / 4334 samples, including the healthy ones. The external data images weren’t used in the validation and placed only in the training part.

After labeling the data I merged it with the source dataset and trained Efficient Net B4 with it.

**Metrics: 0.8999 >> 0.9009**

**4. Final tricks**

By this moment there had left a small list of things which needed to be tried: non-discrete rotations, coarse dropout and standardbs=32.

The first one seems to be stronger than simple discrete 90 degrees rotates that I’m using now. I tried it with a list of models and it had worse performance with the small models and better performance with the larger ones. So I decided to use it for the training of the final ensemble. Parameters: p=0.5, limit=[-180, 180].

I tried coarse dropout with the list of models and with the different parameters as well, starting from the aggressive ones and going to the soft ones. But it was decreasing my score. I understand that the cause might be in the wrong validation and results with the K-fold might be different, but I hadn’t time to train it.

I find standardbs=32 is better than 64 just looking at the learning rate changes plot and later I proved it with the training a couple of models with it - Resnext101 and Efficient Net B4 (results in the Model Zoo report). The final ensemble will be trained using the new value.

**Conclusion:**We successfully implemented additional data, augmentation, made the first k-fold, and greatly increased the accuracy of the model by using the knowledge distillation.

**XI. Final ensembles**

**Parameters changed by that moment:**

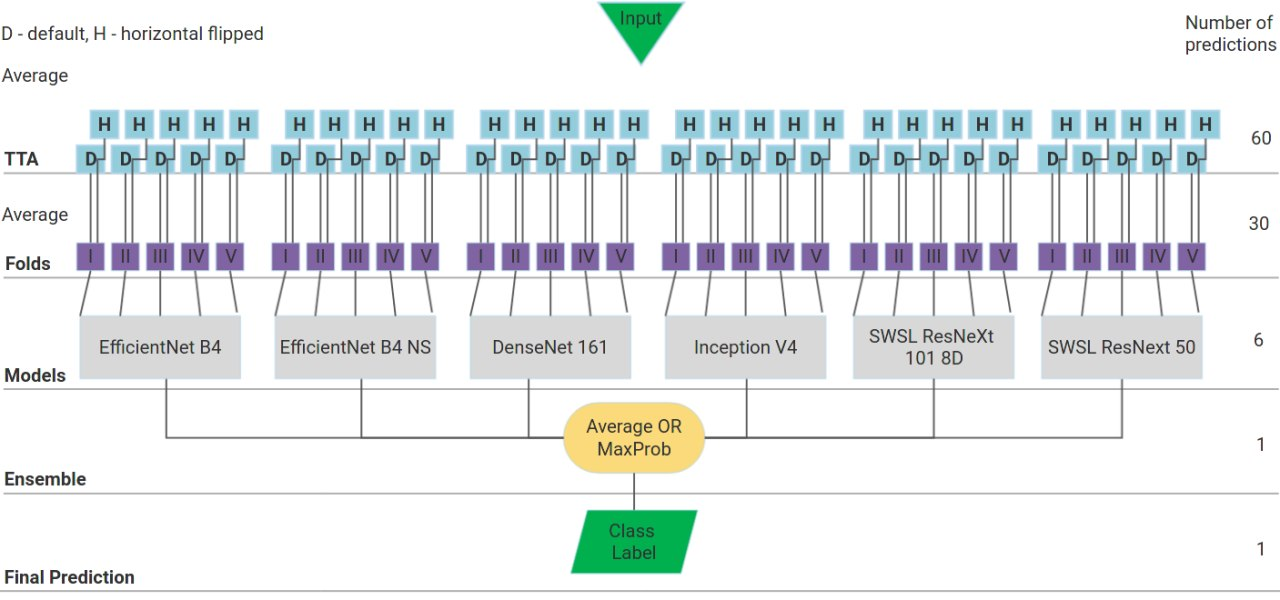
* Standardbs 64 >> 32
* Label smoothing 0.2 >> 0.1
* Discrete rotates >> Non-discrete rotates
* External data used
* Knowledge distillation used

By this moment I’ve found a new way to estimate models. It's a late submission for the 2019 competition. I’ve explored it and noticed that the data and it’s distribution is very close to the ones that we have in the 2020 competition. I’m able to see my private score there as well.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **model** | **Folds** | | | | | **AVG** | **2020**  **LB** | **2019**  **Private**  **LB** |
| **I** | **II** | **III** | **IV** | **V** |
| **EffNetB4** | 0.9067 | 0.9165 | 0.9143 | 0.9033 | 0.9051 | 0.9092 | 0.894 | 0.92932 |
| **ResNeXt 50** | 0.9080 | 0.9134 | 0.9165 | 0.9029 | 0.9160 | 0.9114 | 0.892 | 0.91872 |
| **densenet161** | 0.9085 | 0.9151 | 0.9088 | 0.9001 | 0.9134 | 0.9092 | 0.894 | 0.90989 |
| **SWSL ResNeXt 101 8D** | 0.9129 | 0.9204 | 0.9185 | 0.9048 | 0.9168 | 0.9147 | 0.899 | 0.92800 |
| **SWSL ResNeXt 101 4D** | 0.9085 | 0.9178 | 0.9112 | 0.9030 | 0.9121 | 0.9105 | 0.895 | 0.91475 |
| **vit\_base\_patch16\_384** | 0.8983 | 0.9053 | 0.8677 | 0.8698 | 0.8739 | 0.8830 | - | - |
| **mobilenet\_v2** | 0.9069 | 0.9066 | 0.9067 | 0.8970 | 0.9093 | 0.9053 | 0.894 | 0.91386 |
| **Swsl ResNeXt 50** | 0.9112 | 0.9152 | 0.9163 | 0.9032 | 0.9134 | 0.9119 | 0.896 | 0.92756 |
| **Eff Net B4 (Noisy student)** | 0.9076 | 0.9114 | 0.9154 | 0.9040 | 0.9111 | 0.9099 | 0.890 | 0.92446 |
| **inception\_v4** | 0.9109 | 0.9118 | 0.9104 | 0.9004 | 0.9152 | 0.9097 | 0.897 | 0.91386 |

So here are the final list of models for ensemble and the ensemble scheme:

* SWSL ResNeXt 101 8D
* SWSL ResNeXt 50
* Efficient Net B4
* Efficient Net B4 NS
* Inception V4
* DenseNet 161



**XII. Final Scores**

1) Simple Averaging Ensemble

**2019 Private LB: 0.92712; 2020 Public LB: 0.898; 2020 Private LB: 0.8975**

2) MaxProb Ensemble

Max Probability (or Max Confidence) ensemble allows us to choose the dominating model in the prediction process. In this case, if several models predict different labels, we take the prediction from a model with the biggest confidence.

**2019 Private LB: 0.93197; 2020 Public LB: 0.893; 2020 Private LB: 0.8964**

**Ideas tried:**

* DiffGrad and all another optimizers
* ResNet18
* Mish activation with Ranger optimizer
* Different numbers of epochs (12-50)
* Center crop instead of resize
* Random crop instead of resize
* Bi-Tempered Loss with different parameters
* SCE Loss
* GradualWarmupSchedulerV2 + CosineAnnealingLR
* Small image size (256)
* Big image size (512)
* Different numbers of warm-up epochs
* Different scheduler’s decays
* Ranger optimizer (lr=0.003)
* Label smoothing based on CrossEntropyLoss (custom loss function), smoothing=0.2
* MobileNetV2
* Custom Scheduler (CosineBatchDecayScheduler) + Warm Up
* Horizontal Flip Augmentation
* Vertical Flip Augmentation
* 21 epoch for small pretrained model with turning on scheduler on 4th epoch after warm up
* Progressive image size
* Random rotate augmentation
* ShiftScale Rotate
* HueSaturation Value
* Fmix
* CutMix
* MixUp
* Full 2019 data
* 2019 data except of the most frequent class
* Label smoothing with 0.5 probability instead of 1.0.
* Focal Loss
* Progressive batch size
* Max possible BS on each epoch
* Optical Distortion
* Turning off the blue channel in the image
* Frozen BN layers
* Blur
* Sam optimizer as a wrapper
* ElasticTransform
* Grid Distortion
* RandomSunFlare
* GaussNoise
* Apex
* Mixed precision training
* Gradient accumulation with different parameters
* Model Zoo
* Knowledge distillation
* External data (Mendeley Leaves)
* Coarse dropout
* Average Ensemble
* MaxProb Ensemble
* TTA

**Ideas that worked:**

* Ranger optimizer (lr=0.003)
* Label smoothing based on CrossEntropyLoss (custom loss function), smoothing=0.2
* MobileNetV2 (the best small model)
* Custom Scheduler (CosineBatchDecayScheduler) + Warm Up
* Horizontal Flip Augmentation
* Vertical Flip Augmentation
* Progressive image size
* Random rotate augmentation
* Higher number of epochs with the small model (MobileNetV2) and a bunch of augmentations
* ShiftScale Rotate
* HueSaturation Value
* Fmix
* CutMix
* MixUp
* Frozen BN layers
* Blur
* Mixed precision training
* Gradient accumulation (iters\_to\_accumulate=8)
* Knowledge distillation with label smoothing=0.1
* External data (Mendeley Leaves)
* K-fold
* Ensemble
* 2X TTA

**Neutral Ideas:**

* 2019 data (used only in the train dataset)

**Ideas that didn’t work:**

* DiffGrad and all another optimizers
* ResNet18 (worse than MobileNetV2)
* Mish activation with Ranger optimizer
* Center crop instead of resize
* Random crop instead of resize
* Bi-Tempered Loss
* SCE Loss
* GradualWarmupSchedulerV2 + CosineAnnealingLR
* Label smoothing with 0.5 probability instead of 1.0.
* Focal Loss
* Progressive batch size
* Max possible BS on each epoch
* Optical Distortion
* Turning off the blue channel in the image
* Sam optimizer as a wrapper
* ElasticTransform
* Grid Distortion
* RandomSunFlare
* GaussNoise
* Coarse dropout

**Papers used:**

* [Gradient Centralization: A New Optimization Technique for Deep Neural Networks](https://arxiv.org/abs/2004.01461v2)
* [Robust Bi-Tempered Logistic Loss Based on Bregman Divergences](https://arxiv.org/abs/1906.03361)
* [Delving Deep into Label Smoothing](https://arxiv.org/abs/2011.12562)
* [Bag of Tricks for Image Classification with Convolutional Neural Networks](https://arxiv.org/abs/1812.01187)

**https://www.kaggle.com/c/cassava-leaf-disease-classification**

**Vadim Timakin**

**Maksim Zhdanov**

**Emin Tagiev**

**2020-2021**