# Welcome

Hi everyone ! Welcome to my blog. Here I will just share some tutorials around things that were complicated for me, and for which others R users could be interested. Not surprisingly, lot of this tutorials will involve tensorflow or other deep learning things.

Sometimes things are possible in R, but, since our community is smaller, we don’t have that many resources or tutorials compared to the python community, explaining why it is cubersome to do some particuliar tasks in R, especially when the few tutorials available or interfaces packages start accumulate errors or bugs because they are not used often by an active community.

I am not an expert, so I will try to source at maximum of my codes, or parameters when I can.

There is probably a lack of optimization, but at least it is a working skeleton. If you have suggestion for improvement, comments are welcome 

## About the data

I wrote this code in the first place in the context of the Cassava Leaf Disease Classification, a Kaggle’s competition where the goal was to train a model to identify the disease on leafs of cassava. Here the distillation is made from an Efficientnet0 to an other one.

# What is knowledge distillation

As presented in this discussion thread on kaggle, knowledge distillation is defined as *simply trains another individual model to match the output of an ensemble.* Source. It is in fact slightly more complicated : the second neural net (student) will made predictions on the images, but then, the losses will be a function of its own loss as well as a loss based on the difference between his prediction and the one of its teacher or the ensemble.

This approach allow to compress an ensemble into one model and by then reduce the inference time, or, if trained to match the output of a model, to increase the overall performance of the model. I discover this approach by looking at the top solutions of the Plant Pathology 2020 competition.

I let you go to to this source mentioned aboved to understand how it could potentially works. It does not seems sure, but it seems related to the learning of specific features vs forcing the student to learn “multiple view”, multiple type of feature to detect in the images.

There is off course, no starting material to do it in R. Thanksfully there is a code example on the website of keras. In this example, they create a class of model, a distiller, to make the knowledge distillation. There is, however, one problem : **model are not inheritable in R**. To overcome this problem, I used the code example as a guide, and reproduced the steps by following the approach in this guide for eager executation in keras with R.

**The code is quite hard to understand at first glance**. The reason is, everything is executed in a **single for loop**, since everything is done in eager mode. It did not seemed possible to do it

differently. So there is a lot of variable around to collect metrics during training. If you want to understand the code just remove it from the loop and run it outside of the for loop, before reconstructing the loop around. I did not used tfdataset as shown on the guide for eager execution, so instead of make\_iterator\_one\_shot() and iterator\_get\_next(), here we loop over the train\_generator to produce the batches.

library(tidyverse) library(tensorflow) tf$executing\_eagerly()

[1] TRUE tensorflow::tf\_version() [1] '2.3'

Here I flex with my own version of keras. Basically, it is a fork with application wrapper for the efficient net.

**Disclaimer : I did not write the code for the really handy applications wrappers.** It came from this commit for which the PR is hold until the fully release of tf 2.3, as stated in this PR. I am not sure why the PR is closed.

library(keras)

labels<-read\_csv('train.csv') head(labels)

# A tibble: 6 x 2 image\_id label

<chr> <dbl>

|  |  |
| --- | --- |
| 1 1000015157.jpg | 0 |
| 2 1000201771.jpg | 3 |
| 3 100042118.jpg | 1 |
| 4 1000723321.jpg | 1 |
| 5 1000812911.jpg | 3 |
| 6 1000837476.jpg | 3 |

levels(as.factor(labels$label)) [1] "0" "1" "2" "3" "4"

idx0<-which(labels$label==0) idx1<-which(labels$label==1) idx2<-which(labels$label==2) idx3<-which(labels$label==3) idx4<-which(labels$label==4) labels$CBB<-0

labels$CBSD<-0 labels$CGM<-0 labels$CMD<-0 labels$Healthy<-0 labels$CBB[idx0]<-1 labels$CBSD[idx1]<-1 labels$CGM[idx2]<-1 labels$CMD[idx3]<-1

“Would it have been easier to create a function to convert the labelling ?” You may ask.

labels$Healthy[idx4]<-1

Probably.

#labels$label<-NULL head(labels)

# A tibble: 6 x 7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| image\_id | label | CBB | CBSD | CGM | CMD | Healthy |
| <chr> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> |
| 1 1000015157.jpg | 0 | 1 | 0 | 0 | 0 | 0 |
| 2 1000201771.jpg | 3 | 0 | 0 | 0 | 1 | 0 |
| 3 100042118.jpg | 1 | 0 | 1 | 0 | 0 | 0 |
| 4 1000723321.jpg | 1 | 0 | 1 | 0 | 0 | 0 |
| 5 1000812911.jpg | 3 | 0 | 0 | 0 | 1 | 0 |
| 6 1000837476.jpg | 3 | 0 | 0 | 0 | 1 | 0 |

val\_labels<-read\_csv('validation\_set.csv')

train\_labels<-labels[which(!labels$image\_id %in% val\_labels$image\_id),] table(train\_labels$image\_id %in% val\_labels$image\_id)

FALSE 19256

train\_labels$label<-NULL val\_labels$label<-NULL

head(train\_labels) # A tibble: 6 x 6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| image\_id | CBB | CBSD | CGM | CMD | Healthy |
| <chr> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> |
| 1 1000015157.jpg | 1 | 0 | 0 | 0 | 0 |
| 2 1000201771.jpg | 0 | 0 | 0 | 1 | 0 |
| 3 100042118.jpg | 0 | 1 | 0 | 0 | 0 |
| 4 1000723321.jpg | 0 | 1 | 0 | 0 | 0 |
| 5 1000812911.jpg | 0 | 0 | 0 | 1 | 0 |
| 6 1000837476.jpg | 0 | 0 | 0 | 1 | 0 |

head(val\_labels) # A tibble: 6 x 6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| image\_id | CBB | CBSD | CGM | CMD | Healthy |
| <chr> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> |
| 1 1003442061.jpg | 0 | 0 | 0 | 0 | 1 |
| 2 1004672608.jpg | 0 | 0 | 0 | 1 | 0 |
| 3 1007891044.jpg | 0 | 0 | 0 | 1 | 0 |
| 4 1009845426.jpg | 0 | 0 | 0 | 1 | 0 |
| 5 1010648150.jpg | 0 | 0 | 0 | 1 | 0 |
| 6 1011139244.jpg | 0 | 0 | 0 | 1 | 0 |

image\_path<-'cassava-leaf-disease-classification/train\_images/' #data augmentation

datagen <- image\_data\_generator( rotation\_range = 40,

width\_shift\_range = 0.2,

height\_shift\_range = 0.2,

shear\_range = 0.2,

zoom\_range = 0.5, horizontal\_flip = TRUE, fill\_mode = "reflect"

)

img\_path<-"cassava-leaf-disease-classification/train\_ images/1000015157.jpg"

img <- image\_load(img\_path, target\_size = c(448, 448)) img\_array <- image\_to\_array(img)

img\_array <- array\_reshape(img\_array, c(1, 448, 448, 3)) img\_array<-img\_array/255

# Generated that will flow augmented images augmentation\_generator <- flow\_images\_from\_data(

img\_array,

generator = datagen, batch\_size = 1

)

op <- par(mfrow = c(2, 2), pty = "s", mar = c(1, 0, 1, 0)) for (i in 1:4) {

batch <- generator\_next(augmentation\_generator) plot(as.raster(batch[1,,,]))

}



par(op)

## Data generator

Okay so here is an interresting thing, I will try to compress the code to call a train generator to make it easier to call it.

Why ? Well, apparently a generator does not yield infinite batches, and the for loop of the distiller will stop working without obvious reason at epoch 7, when reaching the end of the validation generator.

When we iterate over it, validation\_generator yeld 8 images and 8 label, until the batch 267, than contains only 5 images (and create the bug when we try to add the loss of the batch to the loss of the epoch. Batch 268 does not exist. So solution seems to recreate on the fly the validation set and restart the iterations.

arg.list <- list(dataframe = val\_labels, directory = image\_path,

class\_mode = "other", x\_col = "image\_id", y\_col = c("CBB","CBSD",

"CGM", "CMD", "Healthy"), 228),

target\_size = c(228,

batch\_size=8)

validation\_generator <- do.call(flow\_images\_from\_dataframe, arg.list) dim(validation\_generator[266][[1]])

[1] 8 228 228 3

dim(validation\_generator[267][[1]]) [1] 5 228 228 3

dim(val\_labels) [1] 2141 6

2141/8

[1] 267.625

train\_generator <- flow\_images\_from\_dataframe(dataframe = train\_labels,

directory = image\_path, generator = datagen, class\_mode = "other", x\_col = "image\_id", y\_col = c("CBB","CBSD",

"CGM", "CMD", "Healthy"), 228),

target\_size = c(228, batch\_size=8)

validation\_generator <- flow\_images\_from\_dataframe(dataframe = val\_labels,

"CGM", "CMD", "Healthy"), 228),

train\_generator

directory = image\_path, class\_mode = "other", x\_col = "image\_id", y\_col = c("CBB","CBSD",

target\_size = c(228, batch\_size=8)

<tensorflow.python.keras.preprocessing.image.DataFrameIterator> conv\_base<-keras::application\_efficientnet\_b0(weights = "imagenet", include\_top = FALSE, input\_shape = c(228, 228, 3))

freeze\_weights(conv\_base)

model <- keras\_model\_sequential() %>% conv\_base %>% layer\_global\_max\_pooling\_2d() %>% layer\_batch\_normalization() %>% layer\_dropout(rate=0.5) %>% layer\_dense(units=5, activation="softmax")

#unfreeze\_weights(model, from = 'block5a\_expand\_conv') unfreeze\_weights(conv\_base, from = 'block5a\_expand\_conv')

model %>% load\_model\_weights\_hdf5("fine\_tuned\_eff\_net\_weights.15.hdf5") summary(model)

Model: "sequential\_2"

Layer (type) Output Shape Param #

======================================================================

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| efficientnetb0 (Functional) | (None, | 8, 8, | 1280) | 4049571 |
| global\_max\_pooling2d\_2 (Global | (None, | 1280) |  | 0 |
| batch\_normalization\_2 (BatchNo | (None, | 1280) |  | 5120 |
| dropout\_2 (Dropout) | (None, | 1280) |  | 0 |

dense\_2 (Dense) (None, 5) 6405

======================================================================

Total params: 4,061,096

Trainable params: 3,707,853

Non-trainable params: 353,243

conv\_base\_student<-keras::application\_efficientnet\_b0(weights = "imagenet", include\_top = FALSE, input\_shape = c(228, 228, 3))

freeze\_weights(conv\_base\_student) student <- keras\_model\_sequential() %>%

conv\_base\_student %>% layer\_global\_max\_pooling\_2d() %>% layer\_batch\_normalization() %>% layer\_dropout(rate=0.5) %>% layer\_dense(units=5, activation="softmax")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| student Model  Model: "sequential\_3" |  | |  | |
| Layer (type) | Output Shape | | Param # | |
| ==============================  efficientnetb0 (Functional) | ======================== (None, 8, 8, 1280) | | | ================ 4049571 |
| global\_max\_pooling2d\_3 (Global | (None, | 1280) |  | 0 |
| batch\_normalization\_3 (BatchNo | (None, | 1280) |  | 5120 |
| dropout\_3 (Dropout) | (None, | 1280) |  | 0 |
| dense\_3 (Dense) | (None, | 5) |  | 6405 |

======================================================================

Total params: 4,061,096

Trainable params: 8,965

Non-trainable params: 4,052,131

## Source code and knowledge distillation

i=1

alpha=0.9 #On\_the\_Efficacy\_of\_Knowledge\_Distillation\_ICCV\_2019 temperature=3

optimizer <- optimizer\_adam()

train\_loss <- tf$keras$metrics$Mean(name='student\_loss') train\_accuracy <- tf$keras$metrics$CategoricalAccuracy(name=' train\_accuracy')

nb\_epoch<-12 nb\_batch<-300 val\_step<-40 train\_loss\_plot<-c() accuracy\_plot<-c()

distilation\_loss\_plot <- c() val\_loss\_plot <- c() val\_accuracy\_plot <- c() count\_epoch<-0

for (epoch in 1:nb\_epoch) {

cat("Epoch: ", epoch, " \n")

# Init metrics train\_loss\_epoch <- 0 accuracies\_on\_epoch <- c() distilation\_loss\_epoch <- 0

val\_loss\_epoch <- 0 val\_accuaries\_on\_epoch <- c()

#Formula to not see the same batch over and over on each epoch #Count epoch instead of epoch

count\_epoch<-count\_epoch+1

idx\_batch <- (1+nb\_batch\*(count\_epoch-1)):(nb\_batch\*count\_epoch) idx\_val\_set <- (1+val\_step\*(count\_epoch-1)):(val\_step\*count\_epoch)

#Dirty solution to restart on a new validation batch generator before reaching the end of the other one

if (as.integer((dim(val\_labels)[1]/8)-1) %in% idx\_val\_set) { count\_epoch<-1

idx\_val\_set <- (1+val\_step\*(count\_epoch-1)):( val\_step\*count\_epoch)

validation\_generator <- do.call(flow\_images\_from\_dataframe, arg.list)

}

#need the same if for train generator

if (as.integer((dim(train\_labels)[1]/8)-1) %in% idx\_batch) { count\_epoch<-1

idx\_batch <- (1+nb\_batch\*(count\_epoch-1)):( nb\_batch\*count\_epoch)

train\_generator <- do.call(flow\_images\_from\_dataframe, arg.list)

}

for (batch in idx\_batch) {

x = train\_generator[batch][[1]] y = train\_generator[batch][[2]] # Forward pass of teacher teacher\_predictions = model(x)

with(tf$GradientTape() %as% tape, { student\_predictions = student(x)

student\_loss = tf$losses$categorical\_crossentropy(y, student\_predictions)

distillation\_loss = tf$losses$categorical\_ crossentropy(tf$nn$softmax(teacher\_predictions/temperature, axis=0L),

tf$nn$softmax(student\_predictions/temperature, axis=0L))

loss = alpha \* student\_loss + (1 - alpha) \* distillation\_loss

})

# Compute gradients

# Variating learning rate :

# optimizer <- optimizer\_adam(lr = 0.0001)

gradients <- tape$gradient(loss, student$trainable\_variables) optimizer$apply\_gradients(purrr::transpose(list(gradients,

student$trainable\_variables)))

#Collect the metrics of the student train\_loss\_epoch <- train\_loss\_epoch + student\_loss distilation\_loss\_epoch <- distilation\_loss\_epoch +

distillation\_loss

accuracy\_on\_batch <- train\_accuracy(y\_true=y, y\_pred=student\_predictions)

accuracies\_on\_epoch <- c(accuracies\_on\_epoch, as.numeric(accuracy\_on\_batch))

}

#Collect info on current epoch and for graphs and cat() train\_loss\_epoch <- mean(as.vector(as.numeric(

train\_loss\_epoch))/nb\_batch)

train\_loss\_plot <- c(train\_loss\_plot, train\_loss\_epoch)

distilation\_loss\_epoch <- mean(as.vector(as.numeric( distilation\_loss\_epoch))/nb\_batch)

distilation\_loss\_plot <- c(distilation\_loss\_plot, distilation\_loss\_epoch)

accuracies\_on\_epoch <- mean(accuracies\_on\_epoch) accuracy\_plot <- c(accuracy\_plot, accuracies\_on\_epoch)

for (step in idx\_val\_set) { # Unpack the data

x = validation\_generator[step][[1]] y = validation\_generator[step][[2]]

# Compute predictions student\_predictions = student(x)

# Calculate the loss

student\_loss = tf$losses$categorical\_crossentropy(y, student\_predictions)

#Collect the metrics of the student

#This line will create a bug of shape when val\_loss end. val\_loss\_epoch <- val\_loss\_epoch + student\_loss

accuracy\_on\_val\_step <- train\_accuracy(y\_true=y, y\_pred=student\_predictions)

val\_accuaries\_on\_epoch <- c(val\_accuaries\_on\_epoch, as.numeric(accuracy\_on\_val\_step))

}

#Collect info on current epoch and for graphs and cat() val\_loss\_epoch <- mean(as.vector(as.numeric(val\_

loss\_epoch))/val\_step)

val\_loss\_plot <- c(val\_loss\_plot, val\_loss\_epoch)

val\_accuaries\_on\_epoch <- mean(val\_accuaries\_on\_epoch) val\_accuracy\_plot <- c(val\_accuracy\_plot, val\_accuaries\_on\_epoch)

#Plotting

cat("Total loss (epoch): ", epoch, ": ", train\_loss\_epoch, "\n") cat("Distillater loss : ", epoch, ": ", distilation\_loss\_epoch,

"\n")

cat("Accuracy (epoch): ", epoch, ": ", accuracies\_on\_epoch, "\n") cat("Val loss : ", epoch, ": ", val\_loss\_epoch, "\n")

cat("Val Accuracy (epoch): ", epoch, ": ", val\_accuaries\_on\_epoch, "\n")

}

Epoch: 1 -

Total loss (epoch): 1 : 2.039012

Distillater loss : 1 : 1.006556

Accuracy (epoch): 1 : 0.5080433

Val loss : 1 : 1.763168

Val Accuracy (epoch): 1 : 0.5439153

Epoch: 2 -

Total loss (epoch): 2 : 1.762901

Distillater loss : 2 : 1.006239

Accuracy (epoch): 2 : 0.5577826

Val loss : 2 : 1.97033

Val Accuracy (epoch): 2 : 0.5661676

Epoch: 3 -

Total loss (epoch): 3 : 1.579749

Distillater loss : 3 : 1.006044

Accuracy (epoch): 3 : 0.5736421

Val loss : 3 : 1.905465

Val Accuracy (epoch): 3 : 0.5780829

Epoch: 4 -

Total loss (epoch): 4 : 1.574974

Distillater loss : 4 : 1.006023

Accuracy (epoch): 4 : 0.5822586

Val loss : 4 : 1.480275

Val Accuracy (epoch): 4 : 0.5850493

Epoch: 5 -

Total loss (epoch): 5 : 1.585655

Distillater loss : 5 : 1.006049

Accuracy (epoch): 5 : 0.5862214

Val loss : 5 : 1.555588

Val Accuracy (epoch): 5 : 0.5880813

Epoch: 6 -

Total loss (epoch): 6 : 1.48109

Distillater loss : 6 : 1.005946

Accuracy (epoch): 6 : 0.591379

Val loss : 6 : 1.34698

Val Accuracy (epoch): 6 : 0.5948141

Epoch: 7 -

Total loss (epoch): 7 : 1.443343

Distillater loss : 7 : 1.005908

Accuracy (epoch): 7 : 0.598381

Val loss : 7 : 2.100892

Val Accuracy (epoch): 7 : 0.5997039

Epoch: 8 -

Total loss (epoch): 8 : 1.505846

Distillater loss : 8 : 1.005823

Accuracy (epoch): 8 : 0.6015843

Val loss : 8 : 1.875012

Val Accuracy (epoch): 8 : 0.6045091

Epoch: 9 -

Total loss (epoch): 9 : 1.459987

Distillater loss : 9 : 1.005817

Accuracy (epoch): 9 : 0.6065652

Val loss : 9 : 2.155602

Val Accuracy (epoch): 9 : 0.6070286

Epoch: 10 -----------

Total loss (epoch): 10 : 1.439232

Distillater loss : 10 : 1.005853

Accuracy (epoch): 10 : 0.607651

Val loss : 10 : 1.204198

Val Accuracy (epoch): 10 : 0.6086346

Epoch: 11 -----------

Total loss (epoch): 11 : 1.46762

Distillater loss : 11 : 1.005828

Accuracy (epoch): 11 : 0.6091381

Val loss : 11 : 1.355449

Val Accuracy (epoch): 11 : 0.6095436

|  |  |  |
| --- | --- | --- |
| Epoch: | 12 ----------- | |
| Total loss (epoch): 12 : | | 1.298911 |
| Distillater loss : 12 : | | 1.005788 |
| Accuracy (epoch): 12 : | | 0.6111491 |

Val loss : 12 : 1.408917

Val Accuracy (epoch): 12 : 0.6121414

## Plotting

total\_loss\_plot<-c()

#instead of collecting them during the training : total\_loss\_plot <- alpha \* train\_loss\_plot + (1 - alpha) \* distilation\_loss\_plot

data <- data.frame("Student\_loss" = train\_loss\_plot,

"Distillation\_loss" = distilation\_loss\_plot, "Total\_loss" = total\_loss\_plot,

"Epoch" = 1:length(train\_loss\_plot), "Val\_loss" = val\_loss\_plot, "Train\_accuracy"= accuracy\_plot, "Val\_accuracy"= val\_accuracy\_plot)

head(data)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Student\_loss | Distillation\_loss | Total\_loss | Epoch | Val\_loss |
| 1 2.039012 | 1.006556 | 1.935766 | 1 | 1.763168 |
| 2 1.762901 | 1.006239 | 1.687235 | 2 | 1.970330 |
| 3 1.579749 | 1.006044 | 1.522379 | 3 | 1.905465 |
| 4 1.574974 | 1.006023 | 1.518078 | 4 | 1.480275 |
| 5 1.585655 | 1.006049 | 1.527694 | 5 | 1.555588 |
| 6 1.481090 | 1.005946 | 1.433575 | 6 | 1.346980 |

|  |  |
| --- | --- |
| Train\_accuracy | Val\_accuracy |
| 1 0.5080433 | 0.5439153 |
| 2 0.5577826 | 0.5661676 |
| 3 0.5736421 | 0.5780829 |
| 4 0.5822586 | 0.5850493 |
| 5 0.5862214 | 0.5880813 |
| 6 0.5913790 | 0.5948141 |

Where total\_loss is alpha \* train\_loss\_plot \* (1 - alpha) \* distilation\_loss\_plot

ggplot(data, aes(Epoch)) + scale\_colour\_manual(values=c(Student\_loss="#F8766D",Val\_

loss="#00BFC4", Distillation\_loss="#DE8C00", Total\_loss="#1aff8c")) + geom\_line(aes(y = Student\_loss, colour = "Student\_loss")) + geom\_line(aes(y = Val\_loss, colour = "Val\_loss")) +

geom\_line(aes(y = Total\_loss, colour = "Total\_loss")) + geom\_line(aes(y = Distillation\_loss, colour = "Distillation\_loss"))



#Validation set ggplot(data, aes(Epoch)) +

geom\_line(aes(y = Train\_accuracy, colour = "Train\_accuracy")) + geom\_line(aes(y = Val\_accuracy, colour = "Val\_accuracy"))



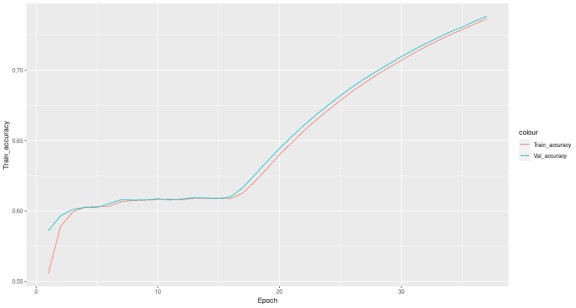
# Fine tuning and conclusion

Is that all ? Well, no. Here we perform knowledge distillation to teach to the head of the student network.

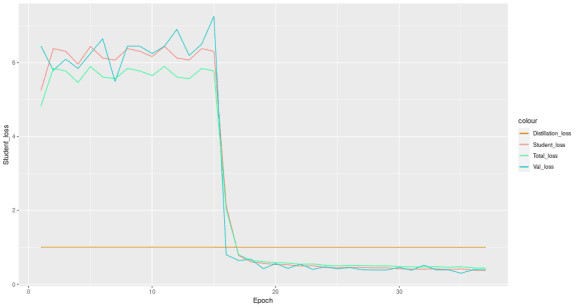
The next step would be to reproduce the knowledge distillation after unfreezing some part of the student, after writing something like :

unfreeze\_weights(conv\_base\_student, from = 'block5a\_expand\_conv')

But I will not bet my small GPU card on this or start a fire in my basement for the sake of the tutorial.



loss



accuracy