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. I used a small size for the images to not blow my GPU, there is an example with fine tuning and a bigger GPU here.

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, an other solution with computer vision and leaf, such as this one.

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**. There is example of inheritance with a R6 for callback, like here, but the models are not a R6 class. 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. I took other code from the tensorflow website for 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.

```
devtools::install_github("Cdk29/keras", dependencies = FALSE)
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"),
                                              target_size = c(228,
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"),
                                              target_size = c(228,
228),
                                              batch_size=8)

validation_generator <- flow_images_from_dataframe(dataframe =
val_labels,
                                              directory = image_path,
                                              class_mode = "other",
                                              x_col = "image_id",
                                              y_col = c("CBB","CBSD",
"CGM", "CMD", "Healthy"),
                                              target_size = c(228,
228),
                                              batch_size=8)
train_generator
<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

Source code for knowledge distillation with Keras : https://keras.io/examples/vision/knowledge_distillation/
Help for eager executation details in R and various usefull code : https://keras.rstudio.com/articles/eager_guide.html
Other source code in R : https://tensorflow.rstudio.com/tutorials/advanced/

I am using an alpha parameter of 0.9 as suggested by this article.

```
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
```

What about global_step = tf.train.get_or_create_global_step() describe here ? It seems to only refers to the number of batches seen by the graph. Source.

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

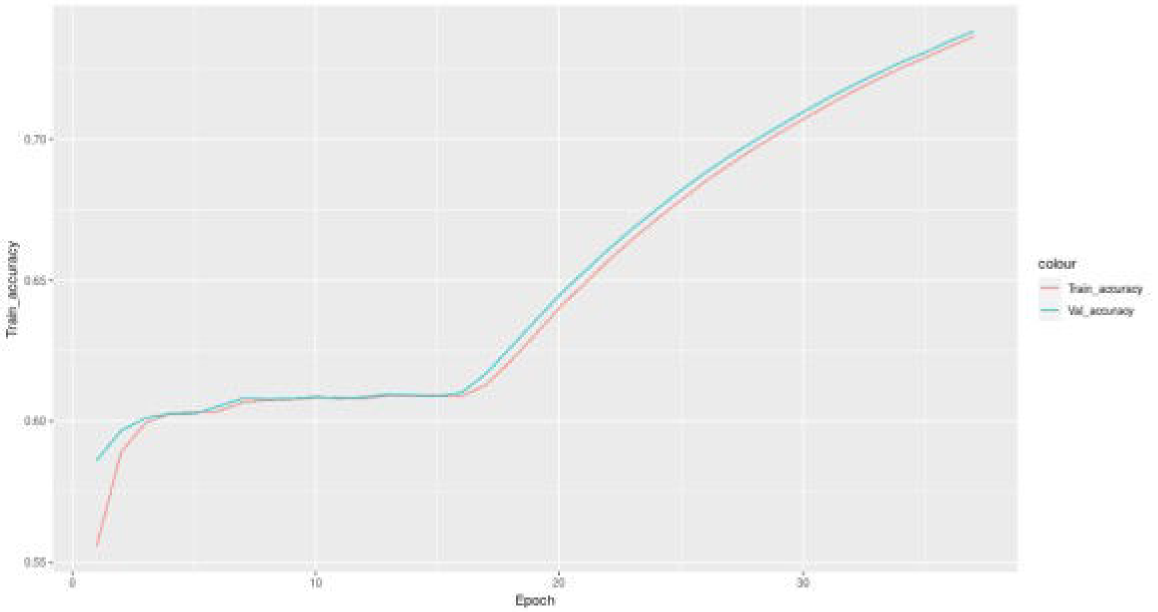
Is that all ? Well, no. Here we perform knowledge distillation to teach to the head of the student network.

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

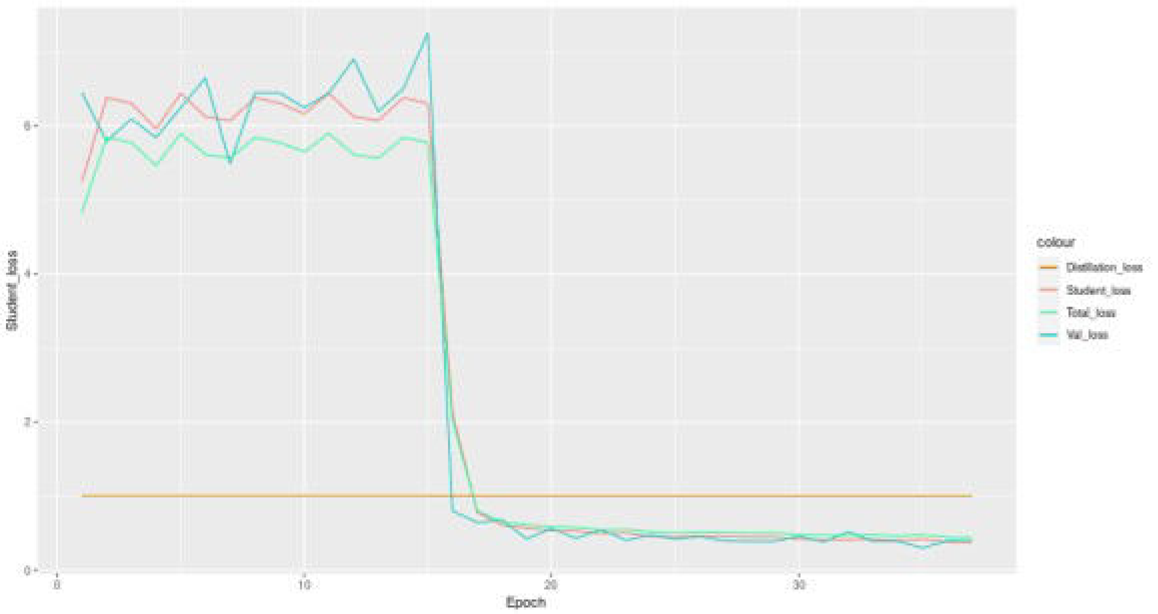
```
unfreeze_weights(conv_base_student, from = 'block5a_expand_conv')
```

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

As I mentioned earlier, I readapted my code from kaggle, where the gpu is much bigger. Take a look if you want to see, but basically the end output looks like this :



loss



accuracy