**The code**

library(keras)

The dataset is the [fruit images dataset from Kaggle](https://www.kaggle.com/moltean/fruits/data). I downloaded it to my computer and unpacked it. Because I don’t want to build a model for all the different fruits, I define a list of fruits (corresponding to the folder names) that I want to include in the model.

I also define a few other parameters in the beginning to make adapting as easy as possible.

# list of fruits to modle

fruit\_list <- c("Kiwi", "Banana", "Plum", "Apricot", "Avocado", "Cocos", "Clementine", "Mandarine", "Orange",

"Limes", "Lemon", "Peach", "Plum", "Raspberry", "Strawberry", "Pineapple", "Pomegranate")

# number of output classes (i.e. fruits)

output\_n <- length(fruit\_list)

# image size to scale down to (original images are 100 x 100 px)

img\_width <- 20

img\_height <- 20

target\_size <- c(img\_width, img\_height)

# RGB = 3 channels

channels <- 3

# path to image folders

train\_image\_files\_path <- "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/"

valid\_image\_files\_path <- "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Validation/"

**Loading images**

The handy image\_data\_generator() and flow\_images\_from\_directory() functions can be used to load images from a directory. If you want to use data augmentation, you can directly define how and in what way you want to augment your images with image\_data\_generator. Here I am not augmenting the data, I only scale the pixel values to fall between 0 and 1.

# optional data augmentation

train\_data\_gen = image\_data\_generator(

rescale = 1/255 #,

#rotation\_range = 40,

#width\_shift\_range = 0.2,

#height\_shift\_range = 0.2,

#shear\_range = 0.2,

#zoom\_range = 0.2,

#horizontal\_flip = TRUE,

#fill\_mode = "nearest"

)

# Validation data shouldn't be augmented! But it should also be scaled.

valid\_data\_gen <- image\_data\_generator(

rescale = 1/255

)

Now we load the images into memory and resize them.

# training images

train\_image\_array\_gen <- flow\_images\_from\_directory(train\_image\_files\_path,

train\_data\_gen,

target\_size = target\_size,

class\_mode = "categorical",

classes = fruit\_list,

seed = 42)

# validation images

valid\_image\_array\_gen <- flow\_images\_from\_directory(valid\_image\_files\_path,

valid\_data\_gen,

target\_size = target\_size,

class\_mode = "categorical",

classes = fruit\_list,

seed = 42)

cat("Number of images per class:")

## Number of images per class:

table(factor(train\_image\_array\_gen$classes))

##

## 0 1 3 4 5 6 7 8 9 10 11 12 13 14 15 16

## 466 490 492 427 490 490 490 479 490 492 492 894 490 492 490 492

cat("\nClass label vs index mapping:\n")

##

## Class label vs index mapping:

train\_image\_array\_gen$class\_indices

## $Lemon

## [1] 10

##

## $Peach

## [1] 11

##

## $Limes

## [1] 9

##

## $Apricot

## [1] 3

##

## $Plum

## [1] 12

##

## $Avocado

## [1] 4

##

## $Strawberry

## [1] 14

##

## $Pineapple

## [1] 15

##

## $Orange

## [1] 8

##

## $Mandarine

## [1] 7

##

## $Banana

## [1] 1

##

## $Clementine

## [1] 6

##

## $Kiwi

## [1] 0

##

## $Cocos

## [1] 5

##

## $Pomegranate

## [1] 16

##

## $Raspberry

## [1] 13

**Define model**

Next, we define the keras model.

# number of training samples

train\_samples <- train\_image\_array\_gen$n

# number of validation samples

valid\_samples <- valid\_image\_array\_gen$n

# define batch size and number of epochs

batch\_size <- 32

epochs <- 10

The model I am using here is a very simple sequential convolutional neural net with the following hidden layers: 2 convolutional layers, one pooling layer and one dense layer.

# initialise model

model <- keras\_model\_sequential()

# add layers

model %>%

layer\_conv\_2d(filter = 32, kernel\_size = c(3,3), padding = "same", input\_shape = c(img\_width, img\_height, channels)) %>%

layer\_activation("relu") %>%

# Second hidden layer

layer\_conv\_2d(filter = 16, kernel\_size = c(3,3), padding = "same") %>%

layer\_activation\_leaky\_relu(0.5) %>%

layer\_batch\_normalization() %>%

# Use max pooling

layer\_max\_pooling\_2d(pool\_size = c(2,2)) %>%

layer\_dropout(0.25) %>%

# Flatten max filtered output into feature vector

# and feed into dense layer

layer\_flatten() %>%

layer\_dense(100) %>%

layer\_activation("relu") %>%

layer\_dropout(0.5) %>%

# Outputs from dense layer are projected onto output layer

layer\_dense(output\_n) %>%

layer\_activation("softmax")

# compile

model %>% compile(

loss = "categorical\_crossentropy",

optimizer = optimizer\_rmsprop(lr = 0.0001, decay = 1e-6),

metrics = "accuracy"

)

Fit the model; because I used image\_data\_generator() and flow\_images\_from\_directory() I am now also using the fit\_generator() to run the training.

# fit

hist <- model %>% fit\_generator(

# training data

train\_image\_array\_gen,

# epochs

steps\_per\_epoch = as.integer(train\_samples / batch\_size),

epochs = epochs,

# validation data

validation\_data = valid\_image\_array\_gen,

validation\_steps = as.integer(valid\_samples / batch\_size),

# print progress

verbose = 2,

callbacks = list(

# save best model after every epoch

callback\_model\_checkpoint("../../data/keras/fruits\_checkpoints.h5", save\_best\_only = TRUE),

# only needed for visualising with TensorBoard

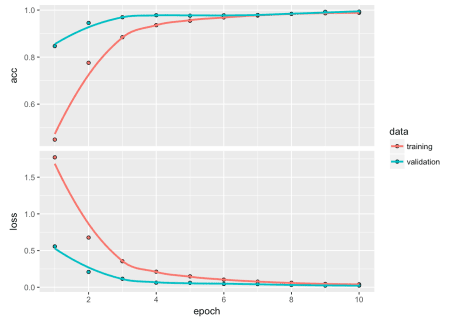
callback\_tensorboard(log\_dir = "../../data/logs/fruits\_logs")

)

)

In RStudio we are seeing the output as an interactive plot in the “Viewer” pane but we can also plot it:

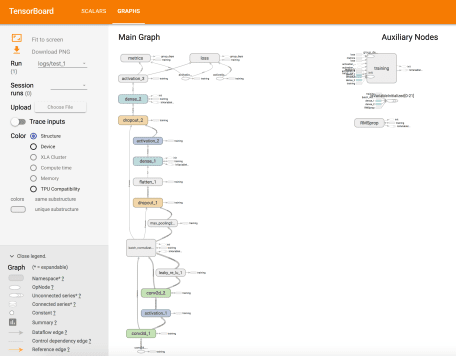
plot(hist)



As we can see, the model is quite accurate on the validation data. However, we need to keep in mind that our images are very uniform, they all have the same white background and show the fruits centered and without anything else in the images. Thus, our model will not work with images that don’t look similar as the ones we trained on (that’s also why we can achieve such good results with such a small neural net).

Finally, I want to have a look at the TensorFlow graph with TensorBoard.

tensorboard("../../data/logs/fruits\_logs")



That’s all there is to it!

Of course, you could now save your model and/or the weights, visualize the hidden layers, run predictions on test data, etc. For now, I’ll leave it at that, though. 

sessionInfo()

## R version 3.5.0 (2018-04-23)

## Platform: x86\_64-apple-darwin15.6.0 (64-bit)

## Running under: macOS High Sierra 10.13.5

##

## Matrix products: default

## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib

## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib

##

## locale:

## [1] de\_DE.UTF-8/de\_DE.UTF-8/de\_DE.UTF-8/C/de\_DE.UTF-8/de\_DE.UTF-8

##

## attached base packages:

## [1] stats graphics grDevices utils datasets methods base

##

## other attached packages:

## [1] keras\_2.1.6

##

## loaded via a namespace (and not attached):

## [1] Rcpp\_0.12.17 compiler\_3.5.0 pillar\_1.2.3 plyr\_1.8.4

## [5] base64enc\_0.1-3 tools\_3.5.0 zeallot\_0.1.0 digest\_0.6.15

## [9] jsonlite\_1.5 evaluate\_0.10.1 tibble\_1.4.2 gtable\_0.2.0

## [13] lattice\_0.20-35 rlang\_0.2.1 Matrix\_1.2-14 yaml\_2.1.19

## [17] blogdown\_0.6 xfun\_0.1 stringr\_1.3.1 knitr\_1.20

## [21] rprojroot\_1.3-2 grid\_3.5.0 reticulate\_1.7 R6\_2.2.2

## [25] rmarkdown\_1.9 bookdown\_0.7 ggplot2\_2.2.1 reshape2\_1.4.3

## [29] magrittr\_1.5 whisker\_0.3-2 backports\_1.1.2 scales\_0.5.0

## [33] tfruns\_1.3 htmltools\_0.3.6 colorspace\_1.3-2 labeling\_0.3

## [37] tensorflow\_1.5 stringi\_1.2.2 lazyeval\_0.2.1 munsell\_0.4.3