What I did not show in that post was how to use the model for making predictions. This, I will do here. But predictions alone are boring, so I’m adding explanations for the predictions using the lime package.

I have already written a few blog posts ([here](https://shirinsplayground.netlify.com/2018/01/looking_beyond_accuracy_to_improve_trust_in_ml/), [here](https://shiring.github.io/machine_learning/2017/04/23/lime) and [here](https://shirinsplayground.netlify.com/2017/12/lime_sketchnotes/)) about LIME and have given talks ([here](https://shirinsplayground.netlify.com/2018/02/m3_2018/) and [here](https://shirinsplayground.netlify.com/2018/04/hh_datascience_meetup_2018_slides/)) about it, too.

Neither of them applies LIME to image classification models, though. And with the new(ish) release from March of [Thomas Lin Pedersen’s lime package](https://cran.r-project.org/web/packages/lime/index.html), lime is now not only on CRAN but it natively supports Keras and image classification models.

Thomas wrote a very nice [article about how to use keras and lime in R](https://www.data-imaginist.com/2018/lime-v0-4-the-kitten-picture-edition/)! Here, I am following this article to use Imagenet (VGG16) to make and explain predictions of fruit images and then I am extending the analysis to [last week’s model](https://shirinsplayground.netlify.com/2018/06/keras_fruits/) and compare it with the pretrained net.

**Loading libraries and models**

library(keras) # for working with neural nets

library(lime) # for explaining models

library(magick) # for preprocessing images

library(ggplot2) # for additional plotting

* Loading the pretrained Imagenet model

model <- application\_vgg16(weights = "imagenet", include\_top = TRUE)

model

## Model

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Layer (type) Output Shape Param #

## ===========================================================================

## input\_1 (InputLayer) (None, 224, 224, 3) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block1\_conv1 (Conv2D) (None, 224, 224, 64) 1792

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block1\_conv2 (Conv2D) (None, 224, 224, 64) 36928

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block1\_pool (MaxPooling2D) (None, 112, 112, 64) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block2\_conv1 (Conv2D) (None, 112, 112, 128) 73856

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block2\_conv2 (Conv2D) (None, 112, 112, 128) 147584

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block2\_pool (MaxPooling2D) (None, 56, 56, 128) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block3\_conv1 (Conv2D) (None, 56, 56, 256) 295168

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block3\_conv2 (Conv2D) (None, 56, 56, 256) 590080

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block3\_conv3 (Conv2D) (None, 56, 56, 256) 590080

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block3\_pool (MaxPooling2D) (None, 28, 28, 256) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block4\_conv1 (Conv2D) (None, 28, 28, 512) 1180160

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block4\_conv2 (Conv2D) (None, 28, 28, 512) 2359808

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block4\_conv3 (Conv2D) (None, 28, 28, 512) 2359808

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block4\_pool (MaxPooling2D) (None, 14, 14, 512) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block5\_conv1 (Conv2D) (None, 14, 14, 512) 2359808

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block5\_conv2 (Conv2D) (None, 14, 14, 512) 2359808

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block5\_conv3 (Conv2D) (None, 14, 14, 512) 2359808

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## block5\_pool (MaxPooling2D) (None, 7, 7, 512) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## flatten (Flatten) (None, 25088) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## fc1 (Dense) (None, 4096) 102764544

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## fc2 (Dense) (None, 4096) 16781312

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## predictions (Dense) (None, 1000) 4097000

## ===========================================================================

## Total params: 138,357,544

## Trainable params: 138,357,544

## Non-trainable params: 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* loading my own model from [last week’s post](https://shirinsplayground.netlify.com/2018/06/keras_fruits/)

model2 <- load\_model\_hdf5(filepath = "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/keras/fruits\_checkpoints.h5")

model2

## Model

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Layer (type) Output Shape Param #

## ===========================================================================

## conv2d\_1 (Conv2D) (None, 20, 20, 32) 896

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## activation\_1 (Activation) (None, 20, 20, 32) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## conv2d\_2 (Conv2D) (None, 20, 20, 16) 4624

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## leaky\_re\_lu\_1 (LeakyReLU) (None, 20, 20, 16) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## batch\_normalization\_1 (BatchNorm (None, 20, 20, 16) 64

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## max\_pooling2d\_1 (MaxPooling2D) (None, 10, 10, 16) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dropout\_1 (Dropout) (None, 10, 10, 16) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## flatten\_1 (Flatten) (None, 1600) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dense\_1 (Dense) (None, 100) 160100

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## activation\_2 (Activation) (None, 100) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dropout\_2 (Dropout) (None, 100) 0

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## dense\_2 (Dense) (None, 16) 1616

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## activation\_3 (Activation) (None, 16) 0

## ===========================================================================

## Total params: 167,300

## Trainable params: 167,268

## Non-trainable params: 32

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Load and prepare images**

Here, I am loading and preprocessing two images of fruits (and yes, I am cheating a bit because I am choosing images where I expect my model to work as they are similar to the training images…).

* Banana

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

img <- image\_read('<https://upload.wikimedia.org/wikipedia/commons/thumb/8/8a/Banana-Single.jpg/272px-Banana-Single.jpg>')

img\_path <- file.path(test\_image\_files\_path, "Banana", 'banana.jpg')

image\_write(img, img\_path)

#plot(as.raster(img))

* Clementine

img2 <- image\_read('<https://cdn.pixabay.com/photo/2010/12/13/09/51/clementine-1792_1280.jpg>')

img\_path2 <- file.path(test\_image\_files\_path, "Clementine", 'clementine.jpg')

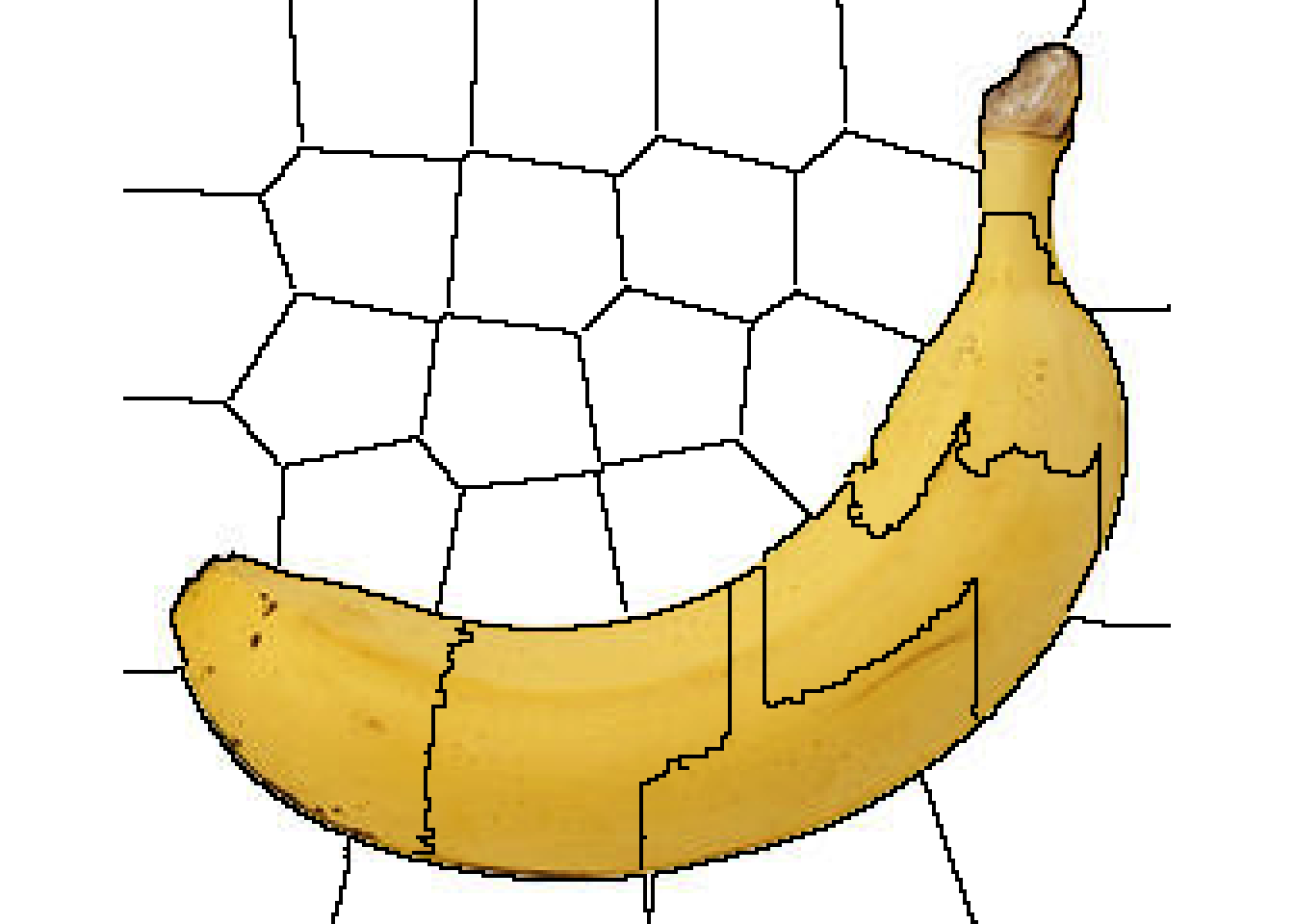
image\_write(img2, img\_path2)

#plot(as.raster(img2))

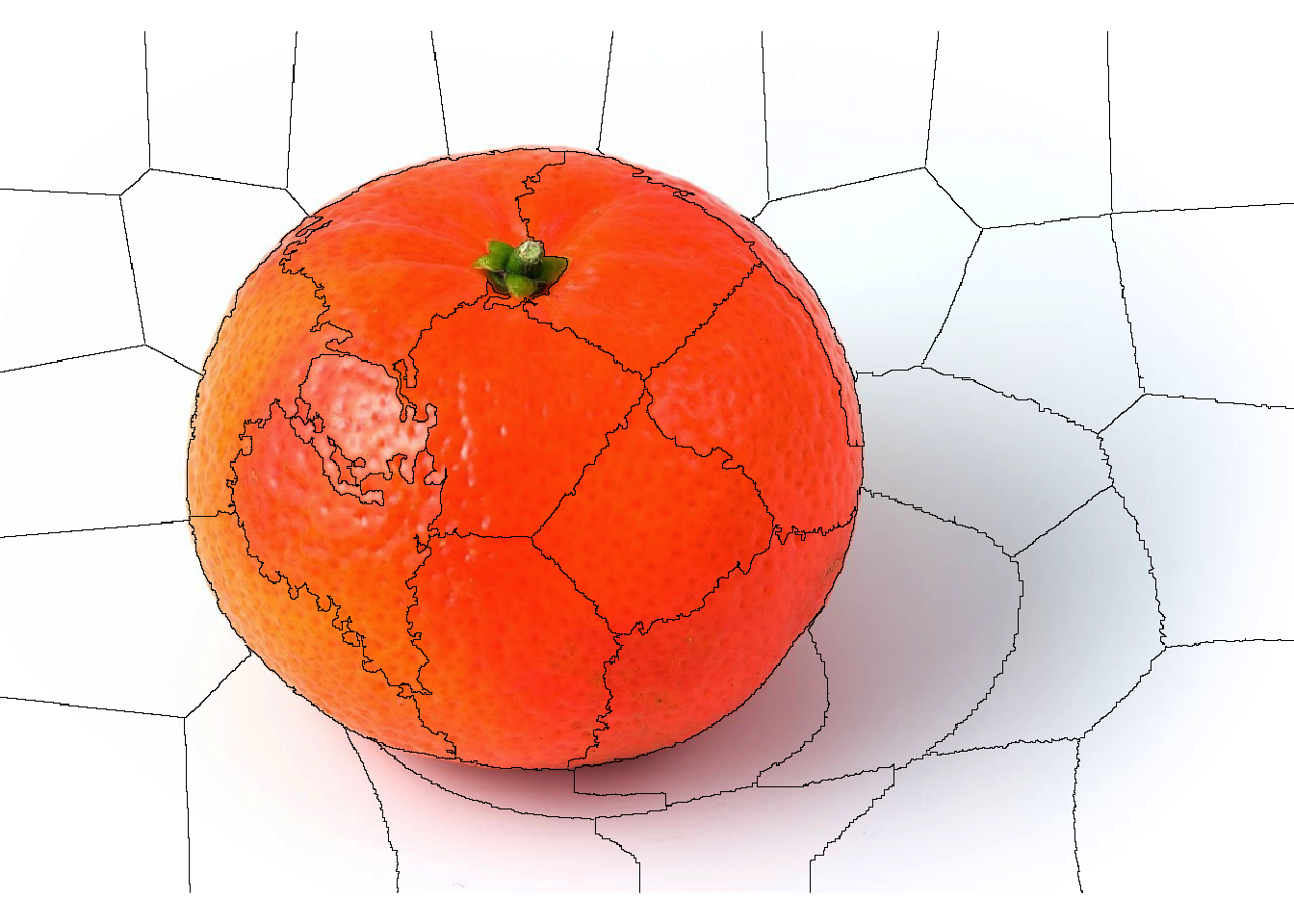
**Superpixels**

The segmentation of an image into superpixels are an important step in generating explanations for image models. It is both important that the segmentation is correct and follows meaningful patterns in the picture, but also that the size/number of superpixels are appropriate. If the important features in the image are chopped into too many segments the permutations will probably damage the picture beyond recognition in almost all cases leading to a poor or failing explanation model. As the size of the object of interest is varying it is impossible to set up hard rules for the number of superpixels to segment into - the larger the object is relative to the size of the image, the fewer superpixels should be generated. Using plot\_superpixels it is possible to evaluate the superpixel parameters before starting the time consuming explanation function. (help(plot\_superpixels))

plot\_superpixels(img\_path, n\_superpixels = 35, weight = 10)



plot\_superpixels(img\_path2, n\_superpixels = 50, weight = 20)



From the superpixel plots we can see that the clementine image has a higher resolution than the banana image.

**Prepare images for Imagenet**

image\_prep <- function(x) {

arrays <- lapply(x, function(path) {

img <- image\_load(path, target\_size = c(224,224))

x <- image\_to\_array(img)

x <- array\_reshape(x, c(1, dim(x)))

x <- imagenet\_preprocess\_input(x)

})

do.call(abind::abind, c(arrays, list(along = 1)))

}

* test predictions

res <- predict(model, image\_prep(c(img\_path, img\_path2)))

imagenet\_decode\_predictions(res)

## [[1]]

## class\_name class\_description score

## 1 n07753592 banana 0.9929747581

## 2 n03532672 hook 0.0013420776

## 3 n07747607 orange 0.0010816186

## 4 n07749582 lemon 0.0010625814

## 5 n07716906 spaghetti\_squash 0.0009176208

##

## [[2]]

## class\_name class\_description score

## 1 n07747607 orange 0.78233224

## 2 n07753592 banana 0.04653566

## 3 n07749582 lemon 0.03868873

## 4 n03134739 croquet\_ball 0.03350329

## 5 n07745940 strawberry 0.01862431

* load labels and train explainer

model\_labels <- readRDS(system.file('extdata', 'imagenet\_labels.rds', package = 'lime'))

explainer <- lime(c(img\_path, img\_path2), as\_classifier(model, model\_labels), image\_prep)

Training the explainer (explain() function) can take pretty long. It will be much faster with the smaller images in my own model but with the bigger Imagenet it takes a few minutes to run.

explanation <- explain(c(img\_path, img\_path2), explainer,

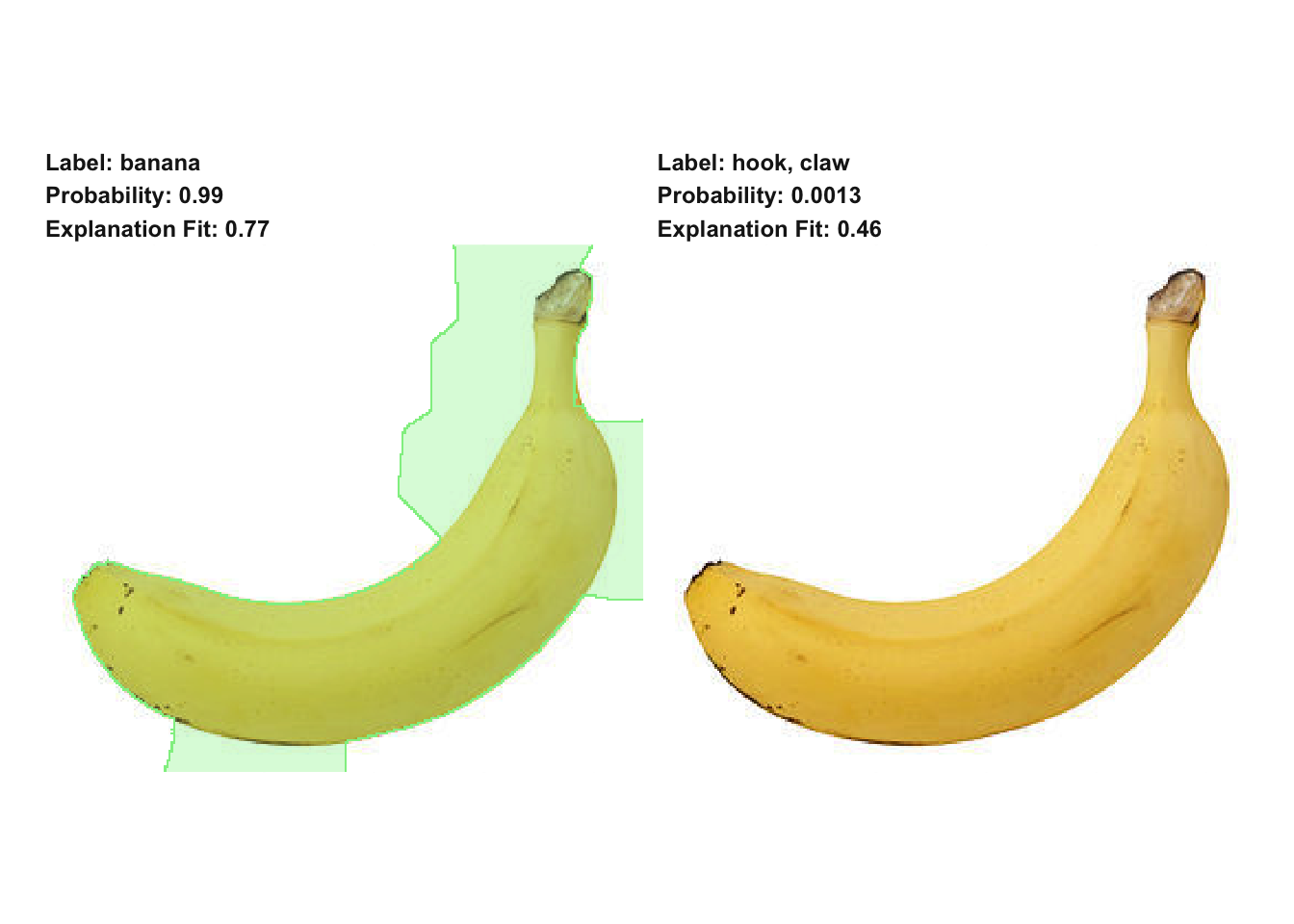
n\_labels = 2, n\_features = 35,

n\_superpixels = 35, weight = 10,

background = "white")

* plot\_image\_explanation() only supports showing one case at a time

plot\_image\_explanation(explanation)



clementine <- explanation[explanation$case == "clementine.jpg",]

plot\_image\_explanation(clementine)



**Prepare images for my own model**

* test predictions (analogous to training and validation images)

test\_datagen <- image\_data\_generator(rescale = 1/255)

test\_generator <- flow\_images\_from\_directory(

test\_image\_files\_path,

test\_datagen,

target\_size = c(20, 20),

class\_mode = 'categorical')

predictions <- as.data.frame(predict\_generator(model2, test\_generator, steps = 1))

load("/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/fruits\_classes\_indices.RData")

fruits\_classes\_indices\_df <- data.frame(indices = unlist(fruits\_classes\_indices))

fruits\_classes\_indices\_df <- fruits\_classes\_indices\_df[order(fruits\_classes\_indices\_df$indices), , drop = FALSE]

colnames(predictions) <- rownames(fruits\_classes\_indices\_df)

t(round(predictions, digits = 2))

## [,1] [,2]

## Kiwi 0 0

## Banana 0 1

## Apricot 0 0

## Avocado 0 0

## Cocos 0 0

## Clementine 1 0

## Mandarine 0 0

## Orange 0 0

## Limes 0 0

## Lemon 0 0

## Peach 0 0

## Plum 0 0

## Raspberry 0 0

## Strawberry 0 0

## Pineapple 0 0

## Pomegranate 0 0

for (i in 1:nrow(predictions)) {

cat(i, ":")

print(unlist(which.max(predictions[i, ])))

}

## 1 :Clementine

## 6

## 2 :Banana

## 2

This seems to be incompatible with lime, though (or if someone knows how it works, please let me know) - so I prepared the images similarly to the Imagenet images.

image\_prep2 <- function(x) {

arrays <- lapply(x, function(path) {

img <- image\_load(path, target\_size = c(20, 20))

x <- image\_to\_array(img)

x <- reticulate::array\_reshape(x, c(1, dim(x)))

x <- x / 255

})

do.call(abind::abind, c(arrays, list(along = 1)))

}

* prepare labels

fruits\_classes\_indices\_l <- rownames(fruits\_classes\_indices\_df)

names(fruits\_classes\_indices\_l) <- unlist(fruits\_classes\_indices)

fruits\_classes\_indices\_l

## 9 10 8 2 11

## "Kiwi" "Banana" "Apricot" "Avocado" "Cocos"

## 3 13 14 7 6

## "Clementine" "Mandarine" "Orange" "Limes" "Lemon"

## 1 5 0 4 15

## "Peach" "Plum" "Raspberry" "Strawberry" "Pineapple"

## 12

## "Pomegranate"

* train explainer

explainer2 <- lime(c(img\_path, img\_path2), as\_classifier(model2, fruits\_classes\_indices\_l), image\_prep2)

explanation2 <- explain(c(img\_path, img\_path2), explainer2,

n\_labels = 1, n\_features = 20,

n\_superpixels = 35, weight = 10,

background = "white")

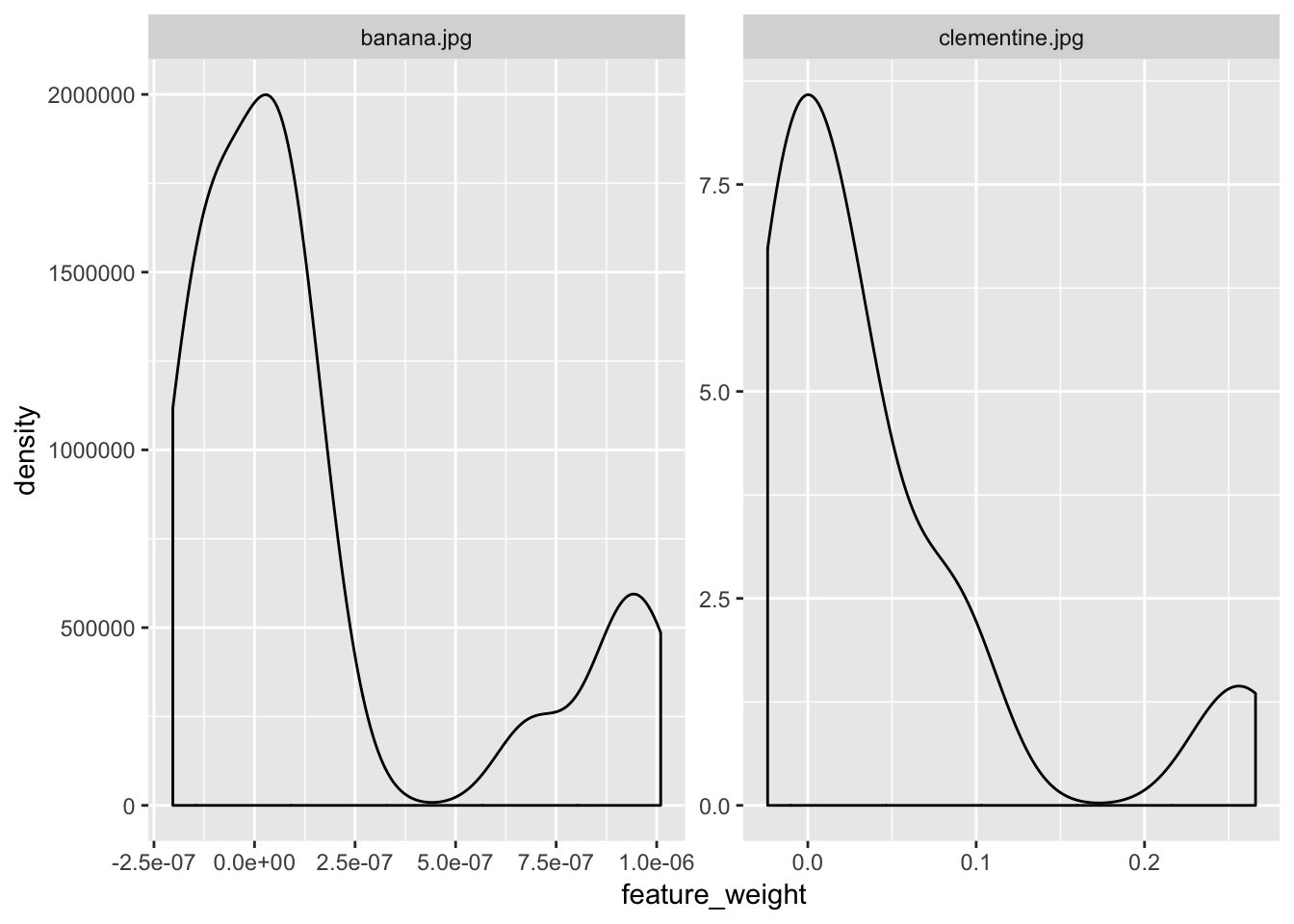
* plot feature weights to find a good threshold for plotting block (see below)

explanation2 %>%

ggplot(aes(x = feature\_weight)) +

facet\_wrap(~ case, scales = "free") +

geom\_density()



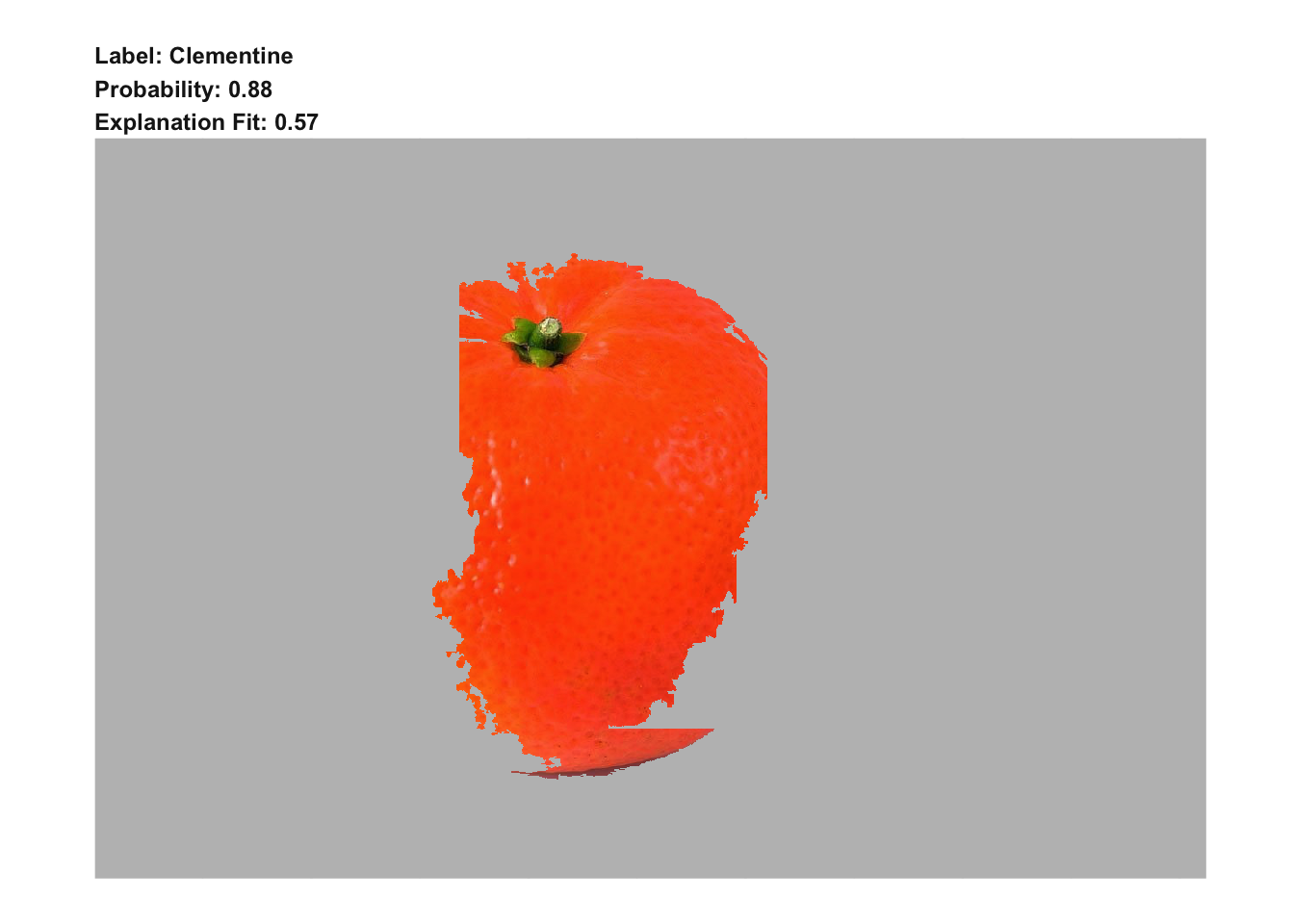
* plot predictions

plot\_image\_explanation(explanation2, display = 'block', threshold = 5e-07)



clementine2 <- explanation2[explanation2$case == "clementine.jpg",]

plot\_image\_explanation(clementine2, display = 'block', threshold = 0.16)



sessionInfo()

## R version 3.5.1 (2018-07-02)

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

## Running under: macOS High Sierra 10.13.6

##

## 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] ggplot2\_3.0.0 magick\_1.9 lime\_0.4.0 keras\_2.1.6

##

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

## [1] stringdist\_0.9.5.1 reticulate\_1.8.0.9003 tidyselect\_0.2.4

## [4] xfun\_0.3 purrr\_0.2.5 lattice\_0.20-35

## [7] colorspace\_1.3-2 htmltools\_0.3.6 yaml\_2.1.19

## [10] base64enc\_0.1-3 rlang\_0.2.1 pillar\_1.2.3

## [13] later\_0.7.3 withr\_2.1.2 glue\_1.2.0

## [16] bindrcpp\_0.2.2 foreach\_1.4.4 plyr\_1.8.4

## [19] bindr\_0.1.1 tensorflow\_1.8 stringr\_1.3.1

## [22] munsell\_0.5.0 blogdown\_0.6 gtable\_0.2.0

## [25] htmlwidgets\_1.2 codetools\_0.2-15 evaluate\_0.10.1

## [28] labeling\_0.3 knitr\_1.20 httpuv\_1.4.4.2

## [31] tfruns\_1.3 curl\_3.2 parallel\_3.5.1

## [34] Rcpp\_0.12.17 xtable\_1.8-2 scales\_0.5.0

## [37] backports\_1.1.2 promises\_1.0.1 jsonlite\_1.5

## [40] abind\_1.4-5 mime\_0.5 digest\_0.6.15

## [43] stringi\_1.2.3 bookdown\_0.7 dplyr\_0.7.6

## [46] shiny\_1.1.0 grid\_3.5.1 rprojroot\_1.3-2

## [49] tools\_3.5.1 magrittr\_1.5 shinythemes\_1.1.1

## [52] lazyeval\_0.2.1 glmnet\_2.0-16 tibble\_1.4.2

## [55] whisker\_0.3-2 pkgconfig\_2.0.1 zeallot\_0.1.0

## [58] Matrix\_1.2-14 gower\_0.1.2 assertthat\_0.2.0

## [61] rmarkdown\_1.10 iterators\_1.0.9 R6\_2.2.2

## [64] compiler\_3.5.1