A while ago, I wrote two blogposts about [image classification with Keras](https://shirinsplayground.netlify.com/2018/06/keras_fruits/) and about [how to use your own models or pretrained models for predictions and using LIME to explain to predictions](https://shirinsplayground.netlify.com/2018/06/keras_fruits_lime/).

Recently, I came across [this blogpost](https://medium.com/@franky07724_57962/using-keras-pre-trained-models-for-feature-extraction-in-image-clustering-a142c6cdf5b1) on using Keras to extract learned features from models and use those to cluster images. It is written in Python, though – so I adapted the code to R. You find the results below.

One use-case for image clustering could be that it can make labelling images easier because – ideally – the clusters would pre-sort your images, so that you only need to go over them quickly and check that they make sense.

**Libraries**

Okay, let’s get started by loading the packages we need.

library(keras)

library(magick) # for preprocessing images

library(tidyverse)

library(imager)

**Pretrained model**

And we load the VGG16 pretrained model but we exclude the laste layers.

model <- application\_vgg16(weights = "imagenet",

include\_top = FALSE)

model

## Model

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

## Layer (type) Output Shape Param #

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Total params: 14,714,688

## Trainable params: 14,714,688

## Non-trainable params: 0

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

**Training images**

Next, I am writting a helper function for reading in images and preprocessing them.

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)))

}

My images are in the following folder:

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

sample\_fruits <- sample(list.dirs(image\_files\_path), 2)

Because running the clustering on all images would take very long, I am randomly sampling 5 image classes.

file\_list <- list.files(sample\_fruits, full.names = TRUE, recursive = TRUE)

head(file\_list)

## [1] "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Apple Red 2/0\_100.jpg"

## [2] "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Apple Red 2/1\_100.jpg"

## [3] "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Apple Red 2/10\_100.jpg"

## [4] "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Apple Red 2/100\_100.jpg"

## [5] "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Apple Red 2/101\_100.jpg"

## [6] "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Apple Red 2/102\_100.jpg"

For each of these images, I am running the predict() function of Keras with the VGG16 model. Because I excluded the last layers of the model, this function will not actually return any class predictions as it would normally do; instead we will get the output of the last layer: block5\_pool (MaxPooling2D).

These, we can use as learned features (or abstractions) of the images. Running this part of the code takes several minutes, so I save the output to a RData file (because I samples randomly, the classes you see below might not be the same as in the sample\_fruits list above).

vgg16\_feature\_list <- data.frame()

for (image in file\_list) {

print(image)

cat("Image", which(file\_list == image), "from", length(file\_list))

vgg16\_feature <- predict(model, image\_prep(image))

flatten <- as.data.frame.table(vgg16\_feature, responseName = "value") %>%

select(value)

flatten <- cbind(image, as.data.frame(t(flatten)))

vgg16\_feature\_list <- rbind(vgg16\_feature\_list, flatten)

}

save(vgg16\_feature\_list, file = "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/vgg16\_feature\_list.RData")

load("/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/vgg16\_feature\_list.RData")

dim(vgg16\_feature\_list)

## [1] 980 25089

**Clustering**

Next, I’m comparing two clustering attempts:

1. clustering original data and
2. clustering first 10 principal components of the data

Here as well, I saved the output to RData because calculation takes some time.

pca <- prcomp(vgg16\_feature\_list[, -1],

center = TRUE,

scale = FALSE)

save(pca, file = "/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/pca.RData")

load("/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/pca.RData")

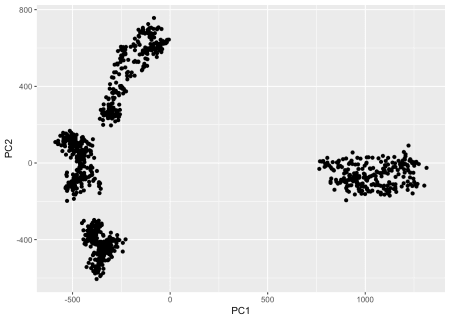
Plotting the first two principal components suggests that the images fall into 4 clusters.

data.frame(PC1 = pca$x[, 1],

PC2 = pca$x[, 2]) %>%

ggplot(aes(x = PC1, y = PC2)) +

geom\_point()



The kMeans function let’s us do [k-Means clustering](https://en.wikipedia.org/wiki/K-means_clustering).

set.seed(50)

cluster\_pca <- kmeans(pca$x[, 1:10], 4)

cluster\_feature <- kmeans(vgg16\_feature\_list[, -1], 4)

Let’s combine the resulting cluster information back with the image information and create a column class (abbreviated with the first three letters).

cluster\_list <- data.frame(cluster\_pca = cluster\_pca$cluster,

cluster\_feature = cluster\_feature$cluster,

vgg16\_feature\_list) %>%

select(cluster\_pca, cluster\_feature, image) %>%

mutate(class = gsub("/Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/", "", image),

class = substr(class, start = 1, stop = 3))

head(cluster\_list)

## cluster\_pca cluster\_feature

## 1 3 2

## 2 3 2

## 3 3 2

## 4 3 2

## 5 3 2

## 6 3 2

## image

## 1 /Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Cocos/100\_100.jpg

## 2 /Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Cocos/101\_100.jpg

## 3 /Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Cocos/102\_100.jpg

## 4 /Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Cocos/103\_100.jpg

## 5 /Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Cocos/104\_100.jpg

## 6 /Users/shiringlander/Documents/Github/DL\_AI/Tutti\_Frutti/fruits-360/Training/Cocos/105\_100.jpg

## class

## 1 Coc

## 2 Coc

## 3 Coc

## 4 Coc

## 5 Coc

## 6 Coc

And let’s count the number of images in each cluster, as well their class.

* PCA:

cluster\_list %>%

count(cluster\_pca, class)

## # A tibble: 4 x 3

## cluster\_pca class n

##

## 1 1 Sal 245

## 2 2 Coc 245

## 3 3 Coc 245

## 4 4 Sal 245

* Original feature map:

cluster\_list %>%

count(cluster\_feature, class)

## # A tibble: 4 x 3

## cluster\_feature class n

##

## 1 1 Sal 245

## 2 2 Coc 490

## 3 3 Sal 157

## 4 4 Sal 88

The classes map pretty clearly to the four clusters from the PCA.

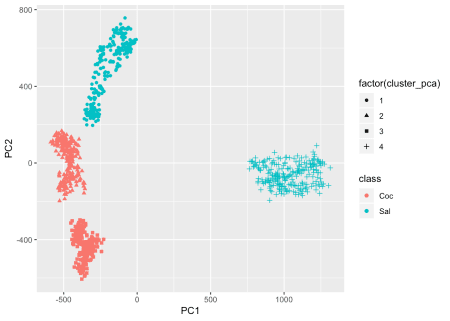
cluster\_list %>%

mutate(PC1 = pca$x[, 1],

PC2 = pca$x[, 2]) %>%

ggplot(aes(x = PC1, y = PC2, color = class, shape = factor(cluster\_pca))) +

geom\_point()



So, let’s plot a few of the images from each cluster so that maybe we’ll be able to see a pattern that explains why our fruits fall into four instead of 2 clusters.

plot\_random\_images <- function(n\_img = 49,

cluster = 1,

rows = 7,

cols = 7) {

cluster\_list\_random\_cl\_1 <- cluster\_list %>%

filter(cluster\_pca == cluster) %>%

sample\_n(n\_img)

graphics::layout(matrix(c(1:n\_img), rows, cols, byrow = TRUE))

for (i in 1:n\_img) {

path <- as.character(cluster\_list\_random\_cl\_1$image[i])

img <- load.image(path)

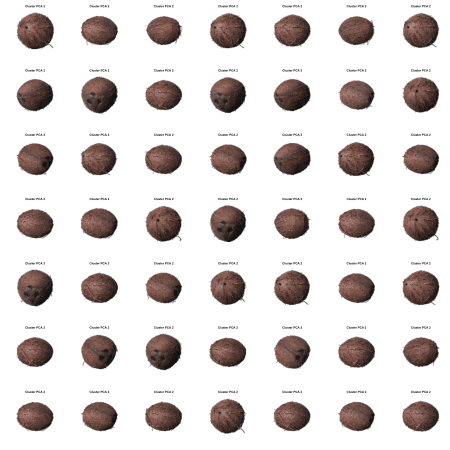
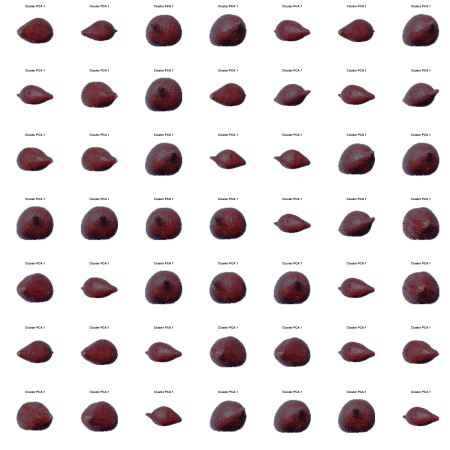
plot(img, axes = FALSE)

title(main = paste("Cluster PCA", cluster))

}

}

sapply(c(1:4), function(x) plot\_random\_images(cluster = x))



## [[1]]

## NULL

##

## [[2]]

## NULL

##

## [[3]]

## NULL

##

## [[4]]

## NULL

Obviously, the clusters reflect the fruits AND the orientation of the fruits. In that way, our clustering represents intuitive patterns in the images that we can understand.

If we didn’t know the classes, labelling our fruits would be much easier now than manually going through each image individually!

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] bindrcpp\_0.2.2 imager\_0.41.1 magrittr\_1.5 forcats\_0.3.0

## [5] stringr\_1.3.1 dplyr\_0.7.6 purrr\_0.2.5 readr\_1.1.1

## [9] tidyr\_0.8.1 tibble\_1.4.2 ggplot2\_3.0.0 tidyverse\_1.2.1

## [13] magick\_2.0 keras\_2.2.0

##

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

## [1] Rcpp\_0.12.19 lubridate\_1.7.4 lattice\_0.20-35 png\_0.1-7

## [5] utf8\_1.1.4 assertthat\_0.2.0 zeallot\_0.1.0 rprojroot\_1.3-2

## [9] digest\_0.6.17 tiff\_0.1-5 R6\_2.3.0 cellranger\_1.1.0

## [13] plyr\_1.8.4 backports\_1.1.2 evaluate\_0.11 httr\_1.3.1

## [17] blogdown\_0.8 pillar\_1.3.0 tfruns\_1.4 rlang\_0.2.2

## [21] lazyeval\_0.2.1 readxl\_1.1.0 rstudioapi\_0.8 whisker\_0.3-2

## [25] bmp\_0.3 Matrix\_1.2-14 reticulate\_1.10 rmarkdown\_1.10

## [29] labeling\_0.3 igraph\_1.2.2 munsell\_0.5.0 broom\_0.5.0

## [33] compiler\_3.5.1 modelr\_0.1.2 xfun\_0.3 pkgconfig\_2.0.2

## [37] base64enc\_0.1-3 tensorflow\_1.9 htmltools\_0.3.6 readbitmap\_0.1.5

## [41] tidyselect\_0.2.4 bookdown\_0.7 fansi\_0.4.0 crayon\_1.3.4

## [45] withr\_2.1.2 grid\_3.5.1 nlme\_3.1-137 jsonlite\_1.5

## [49] gtable\_0.2.0 scales\_1.0.0 cli\_1.0.1 stringi\_1.2.4

## [53] xml2\_1.2.0 tools\_3.5.1 glue\_1.3.0 hms\_0.4.2

## [57] jpeg\_0.1-8 yaml\_2.2.0 colorspace\_1.3-2 rvest\_0.3.2

## [61] knitr\_1.20 bindr\_0.1.1 haven\_1.1.2