

## Labolatorium 06

### Deep Learning – rozpoznawanie zwierząt z obrazków

W tym labolatorium ponownie będę korzystać z języka R oraz programu Rstudio, tym razem jednak będę wykorzystywał bazę obrazów psów i kotów aby nauczyć siec je rozpoznawać. Skorzystałem z bazy dostępnej pod adresem: <https://www.kaggle.com/c/dogs-vs-cats/data>.

Podobnie jak w poprzedniej pracy rozpoczynam od dodania do projektu potrzebnych bibliotek – w tym przypadku jest to **drat**(pozwoli nam wprowadzić pliki obrazów do repozytorium programu) oraz **mxnet**, który odpowiedzialny jest za przetwarzanie obrazów, na podstawie którego sieć będzie się uczyć rozpoznawania.

The screenshot shows the RStudio IDE with the following components:

- Console:** Displays the R version (3.5.2), platform (x86\_64-w64-mingw32/x64), and the successful installation of the 'drat' package from a custom repository. It also shows an error when attempting to load 'mxnet' because it is not found.
- Environment:** Shows the 'Global Environment' with a variable 'cran' of type 'Named chr [1:2]' containing the URL 'https://cran...'. The 'Values' pane is empty.
- Files:** A file explorer showing the project directory 'D:\R\images\_clasificaion'. It contains files like '.Rhistory' (308 B) and 'images\_clasificaion.Rproj' (218 B).

```
R version 3.5.2 (2018-12-20) -- "Eggshell Igloo"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> install.packages("drat", repos = "https://cran.rstudio.com")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/dr
at_0.1.4.zip'
Content type 'application/zip' length 84877 bytes (82 KB)
downloaded 82 KB

package 'drat' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\Łukasz Jaworski\AppData\Local\Temp\RtmpuA30
ns\downloaded_packages
> library(mxnet)
Error in library(mxnet) : nie ma pakietu o nazwie 'mxnet'
> cran <- getOption("repos")
> cran["dm1c"] <- "https://apache-mxnet.s3-accelerate.dualst
ack.amazonaws.com/R/CRAN/"
> options(repos = cran)
> install.packages("mxnet")
also installing the dependencies 'brew', 'colorspace', 'utf8
', 'gtable', 'lazyeval', 'plyr', 'reshape2', 'withr', 'asser
tthat', 'pkgconfig', 'R6', 'tidyselect', 'BH', 'plogr', 'dig
est', 'yaml', 'hms', 'clipr', 'crayon', 'XML', 'Rook', 'labe
ling', 'munsell', 'viridisLite', 'cli', 'fansi', 'pillar', '
ggplot2', 'gridExtra', 'dplyr', 'downloader', 'glue', 'htmlt
ools', 'htmlwidgets', 'igraph', 'influenceR', 'purrr', 'RCol
```

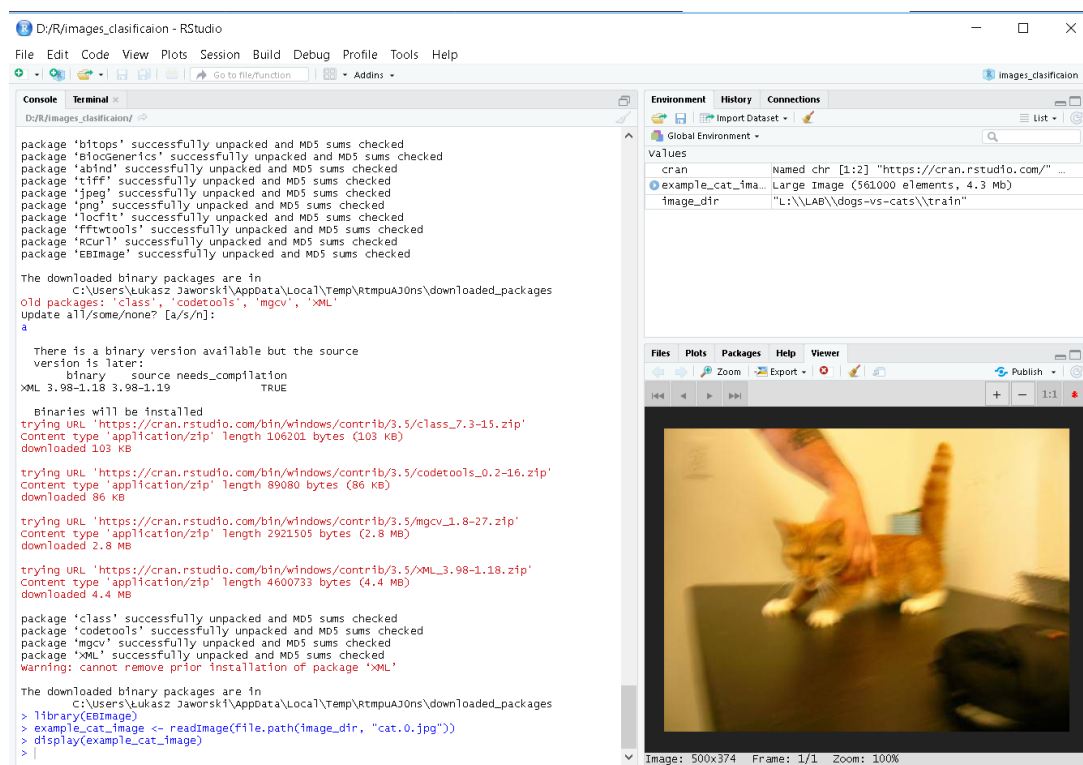
Posiadając obie biblioteki wczytujemy do programu lokalizację plików do nauki i treningu.

```
> library(mxnet)
> image_dir <- "L:\\LAB\\dogs-vs-cats\\train"
> source("https://bioconductor.org/biocLite.R")
trying URL 'https://bioconductor.org/packages/3.7/bioc/bin/windows/contrib/3.5/BiocInstaller_1.30.0.zip'
Content type 'application/zip' length 102191 bytes (99 KB)
downloaded 99 KB

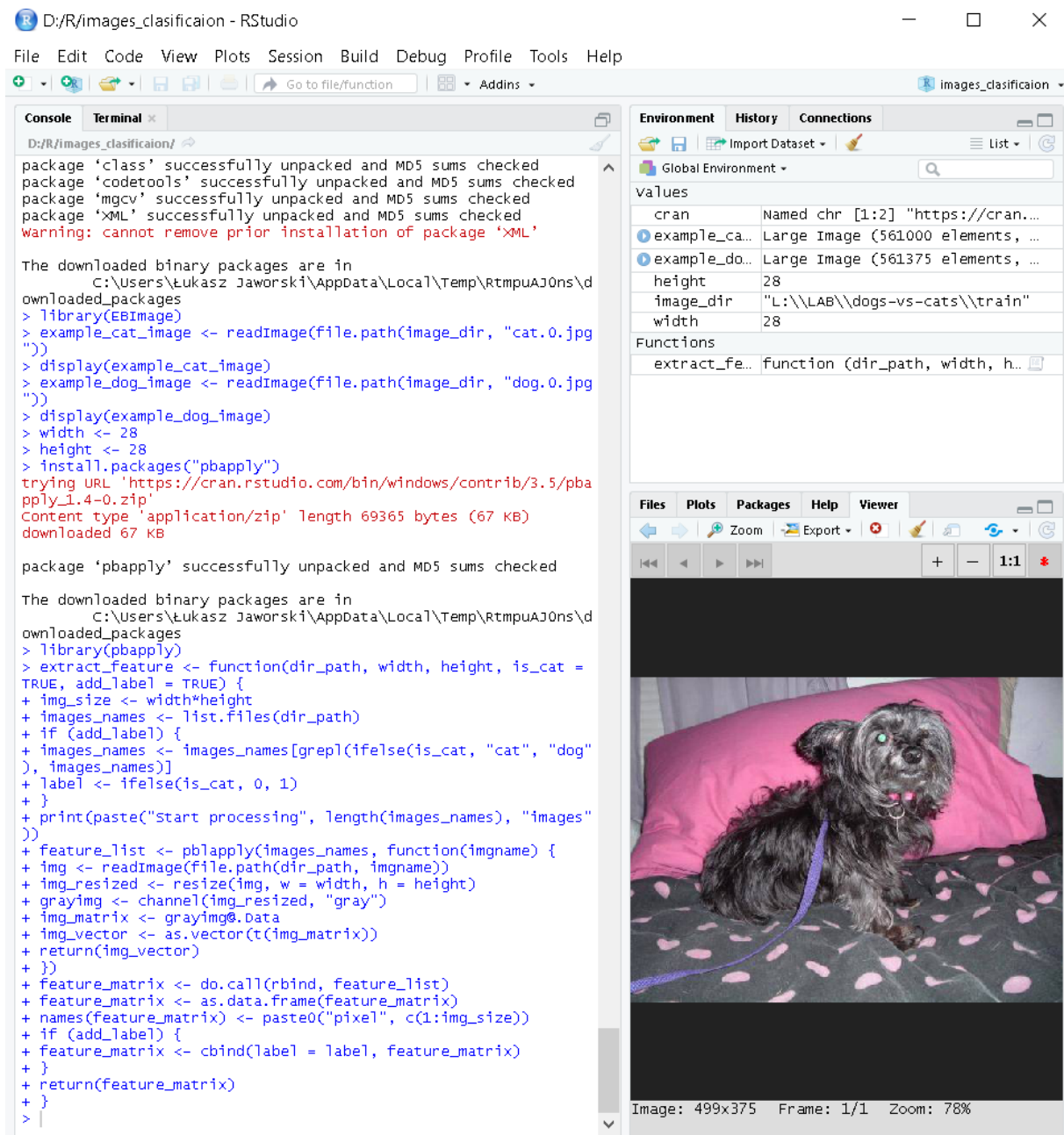
package 'BiocInstaller' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\Lukasz Jaworski\AppData\Local\Temp\RtmpuAJO\
  ns\downloaded_packages
Bioconductor version 3.7 (BiocInstaller 1.30.0), ?biocLite
  for help
A newer version of Bioconductor is available for this
  version of R, ?BiocUpgrade for help
> biocLite("EBImage")
BioC_mirror: https://bioconductor.org
Using Bioconductor 3.7 (BiocInstaller 1.30.0), R 3.5.2
(2018-12-20).
Installing package(s) 'EBImage'
also installing the dependencies 'bitops', 'BiocGenerics', '
  abind', 'tiff', 'jpeg', 'png', 'locfit', 'fftwtools', 'Rcurl
  ',
```

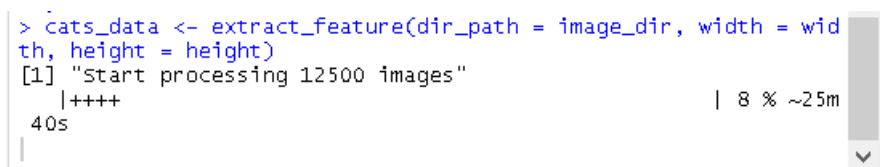
Będzie nam potrzebna jeszcze jedna biblioteka - EBImage, która wspomaga proces analizy plików graficznych. Dzięki niej możemy też załadować plik graficzny w oknie programu.



Na poniższym zrzucie widać operację utworzenia macierzy obrazów z podanej ścieżki do folderu. Folder testowy posiada podział na koty i psy, by program był w stanie określić jakie cechy będą odpowiadające dla różnych rodzajów zwierząt.



Każdy z obrazów zostaje rozbity na mapę w odcieniach szarości, by uzyskać kontury postaci, które będą przetwarzane w programie.



Uzyskane w ten sposób obrazy możemy podzielić na zbiory do trenowania i testowania sieci.

```
-----  
> library(caret)  
Ładowanie wymaganego pakietu: lattice  
Ładowanie wymaganego pakietu: ggplot2  
> complete_set <- rbind(cats_data, dogs_data)  
> training_index <- createDataPartition(complete_set$label, p =  
.9, times = 1)  
> training_index <- unlist(training_index)  
> train_set <- complete_set[training_index,]  
> dim(train_set)  
[1] 22500 785  
> test_set <- complete_set[-training_index,]  
> dim(test_set)  
[1] 2500 785  
> |  
  
> library(caret)  
Ładowanie wymaganego pakietu: lattice  
Ładowanie wymaganego pakietu: ggplot2  
> complete_set <- rbind(cats_data, dogs_data)  
> training_index <- createDataPartition(complete_set$label, p =  
.9, times = 1)  
> training_index <- unlist(training_index)  
> train_set <- complete_set[training_index,]  
> dim(train_set)  
[1] 22500 785  
> |
```

Należy też utworzyć odpowiednie obiekty, które będą dostarczane sieci. Między innymi ustawiamy właściwości macierzy treningowych oraz obrazów w nich zawartych, np. ich rozmiary.

```
> train_data <- data.matrix(train_set)  
> train_x <- t(train_data[, -1])  
> train_y <- train_data[, 1]  
> train_array <- train_x  
> dim(train_array) <- c(28, 28, 1, ncol(train_x))  
> test_data <- data.matrix(test_set)  
> test_x <- t(test_data[, -1])  
> test_y <- test_data[, 1]  
> test_array <- test_x  
> dim(test_array) <- c(28, 28, 1, ncol(test_x))  
> |
```

W tym momencie jesteśmy przygotowani do załadowania wszystkich zapisanych informacji przy użyciu wspomnianej wcześniej biblioteki mxnet. Następnie ustawiamy właściwości sieci neuronowej, jednak nauka rozpoznawania obrazów wymaga innej złożoności sieci niż w poprzednim laboratorium.

Obiekty **conv**, **tanh** i **pool** określają konwulsyjne warstwy sieci, a **fc** – warstwy łączące. Na podstawie ilości filtrów i węzłów nakładanych na obraz określamy jak dokładnie sieć ma sprawdzać własności poszczególnych obrazów. Następnie ustalamy wygląd modelu sieci oraz uruchamiamy trenowanie.

```

> library(mxnet)
> mx_data <- mx.symbol.Variable('data')
> conv_1 <- mx.symbol.Convolution(data = mx_data, kernel = c(5,
5), num_filter = 20)
> tanh_1 <- mx.symbol.Activation(data = conv_1, act_type = "tanh
")
> pool_1 <- mx.symbol.Pooling(data = tanh_1, pool_type = "max",
kernel = c(2, 2), stride = c(2, 2))
> conv_2 <- mx.symbol.Convolution(data = pool_1, kernel = c(5,5)
, num_filter = 50)
> tanh_2 <- mx.symbol.Activation(data = conv_2, act_type = "tanh
")
> pool_2 <- mx.symbol.Pooling(data = tanh_2, pool_type = "max",
kernel = c(2, 2), stride = c(2, 2))
> flat <- mx.symbol.Flatten(data = pool_2)
> fcl_1 <- mx.symbol.FullyConnected(data = flat, num_hidden = 50
0)
> tanh_3 <- mx.symbol.Activation(data = fcl_1, act_type = "tanh"
)
> fcl_2 <- mx.symbol.FullyConnected(data = tanh_3, num_hidden =
2)
> NN_model <- mx.symbol.SoftmaxOutput(data = fcl_2)
> mx.set.seed(100)
> device <- mx.cpu()
> model <- mx.model.FeedForward.create(NN_model, x = train_array
, y = train_y,
+ ctx = device,
+ num.round = 30,
+ array.batch.size = 100,
+ learning.rate = 0.05,
+ momentum = 0.9,
+ wd = 0.00001,
+ eval.metric = mx.metric.accuracy,
+ epoch.end.callback = mx.callback.log.train.metric(100))
Start training with 1 devices
Start training with 1 devices
[1] Train-accuracy=0.495511107842127
[2] Train-accuracy=0.495199996630351
[3] Train-accuracy=0.495199996630351
[4] Train-accuracy=0.495199996630351
[5] Train-accuracy=0.495199996630351
[6] Train-accuracy=0.495644440915849
[7] Train-accuracy=0.498933329714669
[8] Train-accuracy=0.527777774201499
[9] Train-accuracy=0.576533334387673
[10] Train-accuracy=0.615199998087353
[11] Train-accuracy=0.64742222296397
[12] Train-accuracy=0.672000001536475
[13] Train-accuracy=0.689866666838328
[14] Train-accuracy=0.707555555237664
[15] Train-accuracy=0.718977778487735
[16] Train-accuracy=0.732222223811679
[17] Train-accuracy=0.741955555809869
[18] Train-accuracy=0.758488889800178
[19] Train-accuracy=0.76848888847563
[20] Train-accuracy=0.781066664324866
[21] Train-accuracy=0.792488886515299
[22] Train-accuracy=0.805377775298225
[23] Train-accuracy=0.808844442632463
[24] Train-accuracy=0.820933333502875
[25] Train-accuracy=0.823644442293379
[26] Train-accuracy=0.82937777545717
[27] Train-accuracy=0.837822222179837
[28] Train-accuracy=0.836622220675151
[29] Train-accuracy=0.844044443236457
[30] Train-accuracy=0.85777777247959

```

Ostatecznie uzyskaliśmy prawdopodobieństwo odgadnięcia zwierzęcie na poziomie 85% w zbiorze trenującym. Należy jeszcze sprawdzić jak program sobie radzi w losowym zbiorze testowym.

```
> predict_probs <- predict(model, test_array)
> predicted_labels <- max.col(t(predict_probs)) - 1
> table(test_data[, 1], predicted_labels)
      predicted_labels
      0      1
0  740  510
1  329  921
> sum(diag(table(test_data[, 1], predicted_labels)))/2500
[1] 0.6644
> |
```

W tym przypadku osiągnęliśmy prawdopodobieństwo na poziomie 66%. Moglibyśmy osiągnąć wyższy wynik zmieniając własności sieci lub ilości epok, jednak w przypadku przetwarzania plików graficznych każda zmiana w sieci drastycznie wpływa na czas nauki takiej sieci, nie zawsze poprawiając wyniki nauki.