Assignment: Artificial Neural Networks & Deep Learning

Daniel Fuentes, Marc Valenti

5/10/2018

Contents

Data sets: Caltech 101 Description

In the web page: http://www.vision.caltech.edu/Image_Datasets/Caltech101/>
there are pictures of objects belonging to 101 categories. About 40 to 800 images per category. Most categories have about 50 images. Collected in September 2003 by Fei-Fei Li, Marco Andreetto, and Marc 'Aurelio Ranzato. The size of each image is roughly 300 x 200 pixels.

NN architectures

- Select five categories from caltech101 collection. Try to select balanced categories with representative number of images.

```
#only replace the path_images string with where the dataset of pictures from http://www.vision.caltech.

#path_images <- "C:/Users/DanEscario/Desktop/MESIO/Statistical learning/Pr?ctiques/XarxaNeural/101_Obje
path_images <- "~/Desktop/101_ObjectCategories/"

categories <- list.files(path=path_images)

number_images <- rep(NA,length(categories))

for (i in 1:length(categories)){
  number_images[i]<-list.files(path=paste0(path_images,categories[i])) %>%
  length()
}

selected_categories <- categories[number_images%in%(86:88)]
```

After inspecting the number of images of every category, we have ended up choosing *helicopter, kangaroo, menorah, starfish, trilobite*, which have *88, 86, 87, 86, 86* images each.

```
# input_size <- 128 # Width and height of images supplied to the input of a neural network
# epochs <- 30 # The number of epochs
# batch_size <- 16 # Batch size
# orig_width <- 300 # Original image width
# orig_height <- 200 # Height of source images
# train_samples <- 5088 # Size of training sample
# train_index <- sample (1: train_samples, round (train_samples * 0.8)) # 80%
# val_index <- c (1: train_samples) [- train_index]
```

```
# # Folders with pictures
images_dir <- path_images
# masks_dir <- "input / train_masks /"



image_to_grey <- function(image_file,
                          target_width = 32,
                          target_height = 32) {
    img <- image_read(image_file) %>%
      image_convert(colorspace = "gray") %>%
      image_scale(paste0(target_width, "x", target_height, "!"))
    img
}

img2arr <- function(image) {
    result <- as.numeric(image[[1]])
    return(result)
}


#llegeix i escala la imatge
img <- image_to_grey(paste0(path_images,"bonsai/image_0013.jpg"))
#converteix la imatge en un array
img_test <- as.numeric(image_to_grey(paste0(path_images,"bonsai/image_0012.jpg"))[[1]])
```

Implements a classifier fed with the HOG descriptors that learns the category to which a given image belongs. In HOG use 3 cells and 9 orientations. Compare the performance of two NN architectures:V 1. Layer 1: 81 units (input layer), Layer 2: 10 units (hidden layer), Layer 3: 5 units (output layer), 2. Layer 1: 81 units (input layer), Layer 2: 50 units (hidden layer), Layer 3: 25 units (hidden layer), Layer 4: 5 units (output layer).

```
obtain_images_to_HOG <- function(x,
                                 vector_categories,
                                 number_cells=3,
                                 number_orientations=9
                                 ){

  #gets the list of the files in a given directory first.
  imgs <- list.files(paste0(images_dir,vector_categories[x]),
                     pattern = ".jpg",
                     full.names = TRUE)
  #creates an auxiliary matrix where we will place the results.
  #every row is a image. it first contains the label, and then the 1024 vector
  dt_img <- as.data.frame(rep(vector_categories[x],number_images[categories==vector_categories[x]]))
  colnames(dt_img)<-"label"
  dt_img[,2:82]<-0

  #we iterate on all the images
  for(i in 1:length(imgs)) {
    if(length(dim(readImage(imgs[i])))==3){
    img <- readImage(imgs[i]) %>%
      rgb_2gray() %>%
      resizeImage(width = 32,
```

```
                  height = 32,
                  method = "nearest") %>%
      HOG(cells=number_cells,
          orientations=number_orientations)
    dt_img[i,2:82] <- img
    }
    }

  return(dt_img)
}

dt_HOG <- rbind(obtain_images_to_HOG(1,vector_categories=selected_categories),
                obtain_images_to_HOG(2,selected_categories),
                obtain_images_to_HOG(3,selected_categories),
                obtain_images_to_HOG(4,selected_categories),
                obtain_images_to_HOG(5,selected_categories))

trainIndex <- createDataPartition(dt_HOG$label, p=.7, list=F)
dmy <- dummyVars(" ~ label", data = dt_HOG)
dmy_aux <- data.frame(predict(dmy, newdata = dt_HOG))
y_dt_HOG<-dt_HOG$label
dmy_aux <- data.frame(apply(dmy_aux, 2, function(x) (as.numeric(as.character(x)))))
dt_HOG <- cbind(dt_HOG,dmy_aux)
dt_HOG$label<-NULL

resp_var_HOG <- names(select(dmy_aux,starts_with("label")))
xnam <- names(dt_HOG[!names(dt_HOG)%in%resp_var_HOG])

(fmla <- as.formula(paste(paste(resp_var_HOG,collapse="+"), "~", paste(xnam, collapse= "+"))))
```

```
## label.helicopter + label.kangaroo + label.menorah + label.starfish +
##     label.trilobite ~ V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 +
##     V10 + V11 + V12 + V13 + V14 + V15 + V16 + V17 + V18 + V19 +
##     V20 + V21 + V22 + V23 + V24 + V25 + V26 + V27 + V28 + V29 +
##     V30 + V31 + V32 + V33 + V34 + V35 + V36 + V37 + V38 + V39 +
##     V40 + V41 + V42 + V43 + V44 + V45 + V46 + V47 + V48 + V49 +
##     V50 + V51 + V52 + V53 + V54 + V55 + V56 + V57 + V58 + V59 +
##     V60 + V61 + V62 + V63 + V64 + V65 + V66 + V67 + V68 + V69 +
##     V70 + V71 + V72 + V73 + V74 + V75 + V76 + V77 + V78 + V79 +
##     V80 + V81 + V82
```

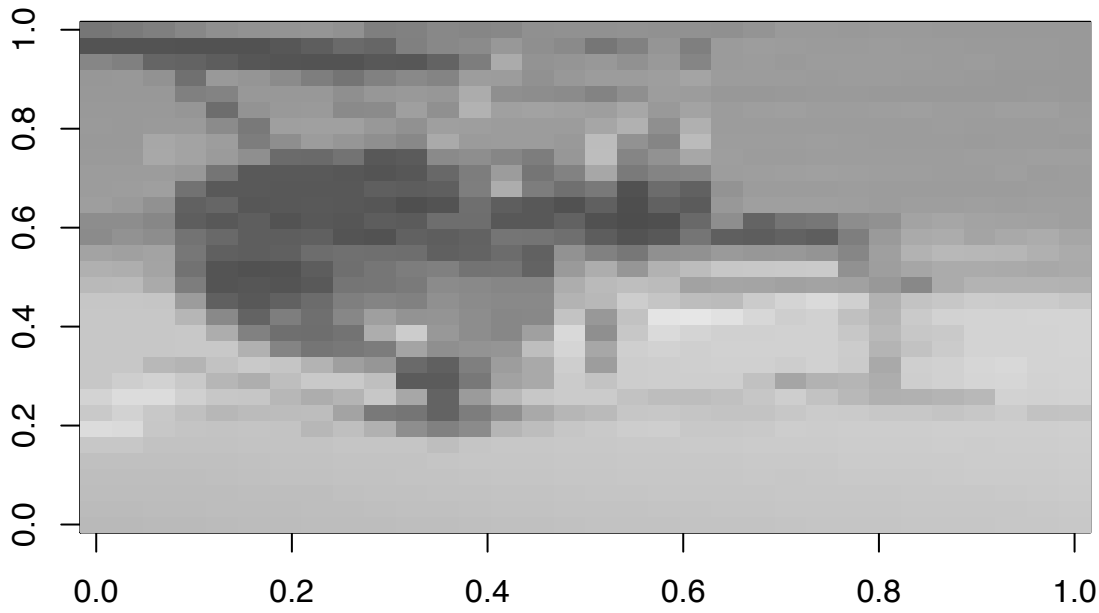
```
dt_HOG_train <- dt_HOG[trainIndex, ]
dt_HOG_test <- dt_HOG[-trainIndex, ]

HOG_grid_10 = expand.grid(layer1 = 10,
                          layer2 = 0,
                          layer3 = 0)

HOG_10_CPU_time <- Sys.time()

HOG.fit_10 = neuralnet(fmla,
                       data = dt_HOG_train,
                       hidden= 10,
                       linear.output=FALSE)
```

```
HOG_10_CPU_time <- Sys.time() - HOG_10_CPU_time

HOG_grid_50 = expand.grid(layer1 = 50,
                          layer2 = 25,
                          layer3=5)

HOG_50_CPU_time <- Sys.time()

HOG.fit_50 = neuralnet(fmla,
                          data = dt_HOG_train,
                          hidden= c(50, 25),
                          linear.output=FALSE)

HOG_50_CPU_time <- Sys.time() - HOG_50_CPU_time


maxidx <- function(row) {
  return(which(row == max(row)))
}
# Plot model and predict
#plotnet(HOG.fit_10) #dona error: "Error in plotnet(HOG.fit): object HOG.fit not found"
train_pred_10 <- compute(HOG.fit_10, dt_HOG_train[,!names(dt_HOG_train) %in% resp_var_HOG])$net.result %
  apply(1, maxidx)

test_pred_10 <- compute(HOG.fit_10, dt_HOG_test[,!names(dt_HOG_test) %in% resp_var_HOG])$net.result %>%
  apply(1, maxidx)

#plotnet(HOG.fit_50)
train_pred_50 <- compute(HOG.fit_50, dt_HOG_train[,!names(dt_HOG_train) %in% resp_var_HOG])$net.result %
  apply(1, maxidx)
test_pred_50 <- compute(HOG.fit_50, dt_HOG_test[,!names(dt_HOG_test) %in% resp_var_HOG])$net.result %>%
  apply(1, maxidx)
```

3. Reshape grayscale images to 32x32. Vectorize images by rows, to get a 1024-vector representation of each image. Implement a NN with two hidden layers that uses such a vectors as input data for learning image categories. Tune the number of neurons in each hidden layer using a grid search procedure.

```
#let's create the dataset first:
#the function obtain_images, would allow us to do this

obtain_images <- function(x,vector_categories){

  #gets the list of the files in a given directory first.
  imgs <- list.files(paste0(images_dir,selected_categories[x]),
                     pattern = ".jpg",
                     full.names = TRUE)
  #creates an auxiliary matrix where we will place the results.
  #every row is a image. it first contains the label, and then the 1024 vector
  dt_img <- as.data.frame(rep(vector_categories[x],number_images[categories==vector_categories[x]]))
  colnames(dt_img)<-"label"
  dt_img[,2:1025]<-0

  #we iterate on all the images
```

```
  for(i in 1:length(imgs)) {
    img <- image_to_grey(imgs[i])
    dt_img[i,2:1025] <- as.vector(img2arr(img))
    }

  return(dt_img)
}

#we create the dataset by row binding the different categories
dt_ex3 <- rbind(obtain_images(1,selected_categories),
            obtain_images(2,selected_categories),
            obtain_images(3,selected_categories),
            obtain_images(4,selected_categories),
            obtain_images(5,selected_categories))

#let's see if that works!
matrix(unlist(dt_ex3[30,-1]), nrow = 32, byrow = TRUE) %>%
  image(col=grey.colors(255))
```




```
image_read(paste0(list.files(paste0(images_dir,selected_categories[1]),
                             pattern = ".jpg",
                             full.names = TRUE))[30])
```

```
#It does! it just acts as a mirror.

#write.csv2(dt,file="data_free_categories_nn.csv")
```

```
trainIndex <- createDataPartition(dt_ex3$label, p=.7, list=F)
dt_ex3_train <- dt_ex3[trainIndex, ]
dt_ex3_test <- dt_ex3[-trainIndex, ]

xnam <- names(dt_ex3_train[-1])
resp_var <- paste0(levels(dt_ex3_train$label),collapse=" + ")
(fmla <- as.formula(paste(resp_var, "~", paste(xnam, collapse= "+"))))
```

```
## helicopter + kangaroo + menorah + starfish + trilobite ~ V2 +
##     V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 + V12 + V13 +
##     V14 + V15 + V16 + V17 + V18 + V19 + V20 + V21 + V22 + V23 +
##     V24 + V25 + V26 + V27 + V28 + V29 + V30 + V31 + V32 + V33 +
##     V34 + V35 + V36 + V37 + V38 + V39 + V40 + V41 + V42 + V43 +
##     V44 + V45 + V46 + V47 + V48 + V49 + V50 + V51 + V52 + V53 +
##     V54 + V55 + V56 + V57 + V58 + V59 + V60 + V61 + V62 + V63 +
##     V64 + V65 + V66 + V67 + V68 + V69 + V70 + V71 + V72 + V73 +
##     V74 + V75 + V76 + V77 + V78 + V79 + V80 + V81 + V82 + V83 +
##     V84 + V85 + V86 + V87 + V88 + V89 + V90 + V91 + V92 + V93 +
##     V94 + V95 + V96 + V97 + V98 + V99 + V100 + V101 + V102 +
##     V103 + V104 + V105 + V106 + V107 + V108 + V109 + V110 + V111 +
##     V112 + V113 + V114 + V115 + V116 + V117 + V118 + V119 + V120 +
##     V121 + V122 + V123 + V124 + V125 + V126 + V127 + V128 + V129 +
##     V130 + V131 + V132 + V133 + V134 + V135 + V136 + V137 + V138 +
##     V139 + V140 + V141 + V142 + V143 + V144 + V145 + V146 + V147 +
##     V148 + V149 + V150 + V151 + V152 + V153 + V154 + V155 + V156 +
##     V157 + V158 + V159 + V160 + V161 + V162 + V163 + V164 + V165 +
##     V166 + V167 + V168 + V169 + V170 + V171 + V172 + V173 + V174 +
##     V175 + V176 + V177 + V178 + V179 + V180 + V181 + V182 + V183 +
##     V184 + V185 + V186 + V187 + V188 + V189 + V190 + V191 + V192 +
##     V193 + V194 + V195 + V196 + V197 + V198 + V199 + V200 + V201 +
##     V202 + V203 + V204 + V205 + V206 + V207 + V208 + V209 + V210 +
```

```
##     V211 + V212 + V213 + V214 + V215 + V216 + V217 + V218 + V219 +
##     V220 + V221 + V222 + V223 + V224 + V225 + V226 + V227 + V228 +
##     V229 + V230 + V231 + V232 + V233 + V234 + V235 + V236 + V237 +
##     V238 + V239 + V240 + V241 + V242 + V243 + V244 + V245 + V246 +
##     V247 + V248 + V249 + V250 + V251 + V252 + V253 + V254 + V255 +
##     V256 + V257 + V258 + V259 + V260 + V261 + V262 + V263 + V264 +
##     V265 + V266 + V267 + V268 + V269 + V270 + V271 + V272 + V273 +
##     V274 + V275 + V276 + V277 + V278 + V279 + V280 + V281 + V282 +
##     V283 + V284 + V285 + V286 + V287 + V288 + V289 + V290 + V291 +
##     V292 + V293 + V294 + V295 + V296 + V297 + V298 + V299 + V300 +
##     V301 + V302 + V303 + V304 + V305 + V306 + V307 + V308 + V309 +
##     V310 + V311 + V312 + V313 + V314 + V315 + V316 + V317 + V318 +
##     V319 + V320 + V321 + V322 + V323 + V324 + V325 + V326 + V327 +
##     V328 + V329 + V330 + V331 + V332 + V333 + V334 + V335 + V336 +
##     V337 + V338 + V339 + V340 + V341 + V342 + V343 + V344 + V345 +
##     V346 + V347 + V348 + V349 + V350 + V351 + V352 + V353 + V354 +
##     V355 + V356 + V357 + V358 + V359 + V360 + V361 + V362 + V363 +
##     V364 + V365 + V366 + V367 + V368 + V369 + V370 + V371 + V372 +
##     V373 + V374 + V375 + V376 + V377 + V378 + V379 + V380 + V381 +
##     V382 + V383 + V384 + V385 + V386 + V387 + V388 + V389 + V390 +
##     V391 + V392 + V393 + V394 + V395 + V396 + V397 + V398 + V399 +
##     V400 + V401 + V402 + V403 + V404 + V405 + V406 + V407 + V408 +
##     V409 + V410 + V411 + V412 + V413 + V414 + V415 + V416 + V417 +
##     V418 + V419 + V420 + V421 + V422 + V423 + V424 + V425 + V426 +
##     V427 + V428 + V429 + V430 + V431 + V432 + V433 + V434 + V435 +
##     V436 + V437 + V438 + V439 + V440 + V441 + V442 + V443 + V444 +
##     V445 + V446 + V447 + V448 + V449 + V450 + V451 + V452 + V453 +
##     V454 + V455 + V456 + V457 + V458 + V459 + V460 + V461 + V462 +
##     V463 + V464 + V465 + V466 + V467 + V468 + V469 + V470 + V471 +
##     V472 + V473 + V474 + V475 + V476 + V477 + V478 + V479 + V480 +
##     V481 + V482 + V483 + V484 + V485 + V486 + V487 + V488 + V489 +
##     V490 + V491 + V492 + V493 + V494 + V495 + V496 + V497 + V498 +
##     V499 + V500 + V501 + V502 + V503 + V504 + V505 + V506 + V507 +
##     V508 + V509 + V510 + V511 + V512 + V513 + V514 + V515 + V516 +
##     V517 + V518 + V519 + V520 + V521 + V522 + V523 + V524 + V525 +
##     V526 + V527 + V528 + V529 + V530 + V531 + V532 + V533 + V534 +
##     V535 + V536 + V537 + V538 + V539 + V540 + V541 + V542 + V543 +
##     V544 + V545 + V546 + V547 + V548 + V549 + V550 + V551 + V552 +
##     V553 + V554 + V555 + V556 + V557 + V558 + V559 + V560 + V561 +
##     V562 + V563 + V564 + V565 + V566 + V567 + V568 + V569 + V570 +
##     V571 + V572 + V573 + V574 + V575 + V576 + V577 + V578 + V579 +
##     V580 + V581 + V582 + V583 + V584 + V585 + V586 + V587 + V588 +
##     V589 + V590 + V591 + V592 + V593 + V594 + V595 + V596 + V597 +
##     V598 + V599 + V600 + V601 + V602 + V603 + V604 + V605 + V606 +
##     V607 + V608 + V609 + V610 + V611 + V612 + V613 + V614 + V615 +
##     V616 + V617 + V618 + V619 + V620 + V621 + V622 + V623 + V624 +
##     V625 + V626 + V627 + V628 + V629 + V630 + V631 + V632 + V633 +
##     V634 + V635 + V636 + V637 + V638 + V639 + V640 + V641 + V642 +
##     V643 + V644 + V645 + V646 + V647 + V648 + V649 + V650 + V651 +
##     V652 + V653 + V654 + V655 + V656 + V657 + V658 + V659 + V660 +
##     V661 + V662 + V663 + V664 + V665 + V666 + V667 + V668 + V669 +
##     V670 + V671 + V672 + V673 + V674 + V675 + V676 + V677 + V678 +
##     V679 + V680 + V681 + V682 + V683 + V684 + V685 + V686 + V687 +
##     V688 + V689 + V690 + V691 + V692 + V693 + V694 + V695 + V696 +
```

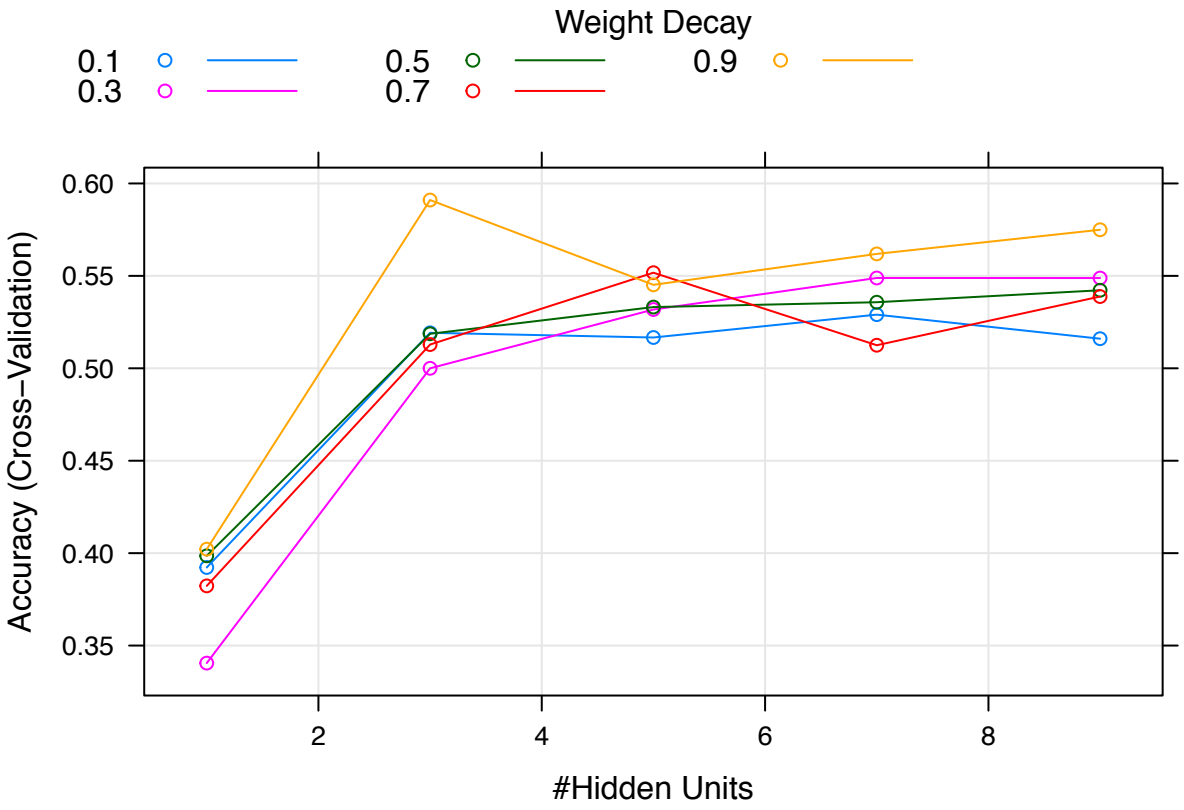
```
##     V697 + V698 + V699 + V700 + V701 + V702 + V703 + V704 + V705 +
##     V706 + V707 + V708 + V709 + V710 + V711 + V712 + V713 + V714 +
##     V715 + V716 + V717 + V718 + V719 + V720 + V721 + V722 + V723 +
##     V724 + V725 + V726 + V727 + V728 + V729 + V730 + V731 + V732 +
##     V733 + V734 + V735 + V736 + V737 + V738 + V739 + V740 + V741 +
##     V742 + V743 + V744 + V745 + V746 + V747 + V748 + V749 + V750 +
##     V751 + V752 + V753 + V754 + V755 + V756 + V757 + V758 + V759 +
##     V760 + V761 + V762 + V763 + V764 + V765 + V766 + V767 + V768 +
##     V769 + V770 + V771 + V772 + V773 + V774 + V775 + V776 + V777 +
##     V778 + V779 + V780 + V781 + V782 + V783 + V784 + V785 + V786 +
##     V787 + V788 + V789 + V790 + V791 + V792 + V793 + V794 + V795 +
##     V796 + V797 + V798 + V799 + V800 + V801 + V802 + V803 + V804 +
##     V805 + V806 + V807 + V808 + V809 + V810 + V811 + V812 + V813 +
##     V814 + V815 + V816 + V817 + V818 + V819 + V820 + V821 + V822 +
##     V823 + V824 + V825 + V826 + V827 + V828 + V829 + V830 + V831 +
##     V832 + V833 + V834 + V835 + V836 + V837 + V838 + V839 + V840 +
##     V841 + V842 + V843 + V844 + V845 + V846 + V847 + V848 + V849 +
##     V850 + V851 + V852 + V853 + V854 + V855 + V856 + V857 + V858 +
##     V859 + V860 + V861 + V862 + V863 + V864 + V865 + V866 + V867 +
##     V868 + V869 + V870 + V871 + V872 + V873 + V874 + V875 + V876 +
##     V877 + V878 + V879 + V880 + V881 + V882 + V883 + V884 + V885 +
##     V886 + V887 + V888 + V889 + V890 + V891 + V892 + V893 + V894 +
##     V895 + V896 + V897 + V898 + V899 + V900 + V901 + V902 + V903 +
##     V904 + V905 + V906 + V907 + V908 + V909 + V910 + V911 + V912 +
##     V913 + V914 + V915 + V916 + V917 + V918 + V919 + V920 + V921 +
##     V922 + V923 + V924 + V925 + V926 + V927 + V928 + V929 + V930 +
##     V931 + V932 + V933 + V934 + V935 + V936 + V937 + V938 + V939 +
##     V940 + V941 + V942 + V943 + V944 + V945 + V946 + V947 + V948 +
##     V949 + V950 + V951 + V952 + V953 + V954 + V955 + V956 + V957 +
##     V958 + V959 + V960 + V961 + V962 + V963 + V964 + V965 + V966 +
##     V967 + V968 + V969 + V970 + V971 + V972 + V973 + V974 + V975 +
##     V976 + V977 + V978 + V979 + V980 + V981 + V982 + V983 + V984 +
##     V985 + V986 + V987 + V988 + V989 + V990 + V991 + V992 + V993 +
##     V994 + V995 + V996 + V997 + V998 + V999 + V1000 + V1001 +
##     V1002 + V1003 + V1004 + V1005 + V1006 + V1007 + V1008 + V1009 +
##     V1010 + V1011 + V1012 + V1013 + V1014 + V1015 + V1016 + V1017 +
##     V1018 + V1019 + V1020 + V1021 + V1022 + V1023 + V1024 + V1025
```

```
my.grid <- expand.grid(size=seq(from=1,to=10,by=2),
                       decay=seq(from=0.1,to=1,by=0.2))
ex3_grid_search_time<-Sys.time()
ex3.fit <- train(label ~ .,
                 data = dt_ex3_train,
                 method = "nnet",
                 tuneGrid = my.grid,
                 verbose=FALSE,
                 preProc =  c('center', 'scale', 'knnImpute', 'pca'),
                 trControl = trainControl(method = "cv",
                                          verboseIter = FALSE,
                                          number = 5,
                                          returnData = FALSE)
                 # ,MaxNWts=1036
                 )
```

```
## # weights:  85
```

```
## iter  50 value 173.545654
## iter  60 value 162.719399
## iter  70 value 155.870170
## iter  80 value 148.626149
## iter  90 value 145.409689
## iter 100 value 144.297627
## final  value 144.297627
## stopped after 100 iterations
## # weights:  716
## initial  value 595.572725
## iter  10 value 331.362170
## iter  20 value 250.856571
## iter  30 value 195.719995
## iter  40 value 163.772559
## iter  50 value 151.357016
## iter  60 value 143.996295
## iter  70 value 138.247261
## iter  80 value 134.673976
## iter  90 value 133.142627
## iter 100 value 132.064214
## final  value 132.064214
## stopped after 100 iterations
## # weights:  621
## initial  value 647.664528
## iter  10 value 368.687520
## iter  20 value 304.269749
## iter  30 value 262.423579
## iter  40 value 217.116086
## iter  50 value 184.993070
## iter  60 value 164.846343
## iter  70 value 154.594602
## iter  80 value 143.777923
## iter  90 value 139.169516
## iter 100 value 136.392598
## final  value 136.392598
## stopped after 100 iterations
```

```
ex3_grid_search_time<-Sys.time() - ex3_grid_search_time

plot(ex3.fit)
```



Doing the grid search procedure with a 5 fold CV has taken 1.5308816512. The CPU time in this case will be a function of how big our grid.search is and how many folds the CV has. Therefore, we now estimate the new CPU time with the best tune: 7 as size and 0.5 as decay.

```
my.grid <- expand.grid(size=ex3.fit$bestTune$size,
                       decay=ex3.fit$bestTune$decay)

ex3_fit_time<-Sys.time()
ex3.fit <- train(label ~ .,
                 data = dt_ex3_train,
                 method = "nnet",
                 tuneGrid = my.grid,
                 verbose=FALSE,
                 preProc =  c('center', 'scale', 'knnImpute', 'pca'),
                 trControl = trainControl(method = "cv",
                                          verboseIter = FALSE,
                                          number = 10,
                                          returnData = FALSE)
                 # ,MaxNWts=1036
                 )
```

```
## # weights:  607
## initial  value 559.015309
## iter  10 value 330.779221
## iter  20 value 281.536276
## iter  30 value 229.926281
## iter  40 value 182.780938
```

```
## iter  50 value 155.450435
## iter  60 value 145.249051
## iter  70 value 131.244724
## iter  80 value 124.876203
## iter  90 value 121.602509
## iter 100 value 120.441214
## final  value 120.441214
## stopped after 100 iterations
## # weights:  593
## initial  value 561.952372
## iter  10 value 318.815677
## iter  20 value 262.843930
## iter  30 value 224.132340
## iter  40 value 194.248426
## iter  50 value 177.037208
## iter  60 value 165.775516
## iter  70 value 155.380407
## iter  80 value 150.097929
## iter  90 value 146.006412
## iter 100 value 143.853871
## final  value 143.853871
## stopped after 100 iterations
## # weights:  579
## initial  value 619.190388
## iter  10 value 353.483766
## iter  20 value 297.533552
## iter  30 value 252.116005
## iter  40 value 213.262950
## iter  50 value 197.084655
## iter  60 value 182.811745
## iter  70 value 164.731693
## iter  80 value 152.195225
## iter  90 value 143.907690
## iter 100 value 138.547114
## final  value 138.547114
## stopped after 100 iterations
## # weights:  586
## initial  value 523.024303
## iter  10 value 323.177873
## iter  20 value 266.970861
## iter  30 value 216.298884
## iter  40 value 181.066403
## iter  50 value 156.439151
## iter  60 value 142.749586
## iter  70 value 134.291925
## iter  80 value 127.016980
## iter  90 value 123.074999
## iter 100 value 120.944222
## final  value 120.944222
## stopped after 100 iterations
## # weights:  600
## initial  value 576.970862
## iter  10 value 389.825347
## iter  20 value 319.780232
```

```
## iter  30 value 268.733501
## iter  40 value 220.299625
## iter  50 value 188.195881
## iter  60 value 160.117232
## iter  70 value 146.437586
## iter  80 value 135.667292
## iter  90 value 131.264236
## iter 100 value 129.446844
## final  value 129.446844
## stopped after 100 iterations
## # weights:  600
## initial  value 549.236699
## iter  10 value 321.680284
## iter  20 value 255.093438
## iter  30 value 211.477523
## iter  40 value 174.774963
## iter  50 value 150.439103
## iter  60 value 141.708056
## iter  70 value 130.085627
## iter  80 value 125.884751
## iter  90 value 123.516667
## iter 100 value 121.999893
## final  value 121.999893
## stopped after 100 iterations
## # weights:  586
## initial  value 557.140801
## iter  10 value 342.120900
## iter  20 value 284.489204
## iter  30 value 242.641977
## iter  40 value 209.161666
## iter  50 value 183.686934
## iter  60 value 164.344607
## iter  70 value 152.345142
## iter  80 value 142.383273
## iter  90 value 134.208212
## iter 100 value 131.313107
## final  value 131.313107
## stopped after 100 iterations
## # weights:  593
## initial  value 501.967916
## iter  10 value 315.481536
## iter  20 value 271.561921
## iter  30 value 220.536532
## iter  40 value 191.682009
## iter  50 value 166.259417
## iter  60 value 152.692319
## iter  70 value 139.912162
## iter  80 value 130.452532
## iter  90 value 126.773669
## iter 100 value 125.174035
## final  value 125.174035
## stopped after 100 iterations
## # weights:  600
## initial  value 515.652097
```

```
## iter  10 value 331.188144
## iter  20 value 287.206404
## iter  30 value 233.344038
## iter  40 value 193.987888
## iter  50 value 171.576418
## iter  60 value 154.184445
## iter  70 value 140.245486
## iter  80 value 127.058210
## iter  90 value 120.888641
## iter 100 value 118.134358
## final  value 118.134358
## stopped after 100 iterations
## # weights:  593
## initial  value 538.039721
## iter  10 value 338.931176
## iter  20 value 286.719812
## iter  30 value 235.833928
## iter  40 value 191.486994
## iter  50 value 168.345129
## iter  60 value 156.983503
## iter  70 value 144.102982
## iter  80 value 136.325986
## iter  90 value 133.412310
## iter 100 value 130.447274
## final  value 130.447274
## stopped after 100 iterations
## # weights:  621
## initial  value 580.892030
## iter  10 value 355.620986
## iter  20 value 304.526695
## iter  30 value 255.737697
## iter  40 value 203.464396
## iter  50 value 183.018423
## iter  60 value 167.813138
## iter  70 value 158.789683
## iter  80 value 151.289120
## iter  90 value 148.549402
## iter 100 value 145.918786
## final  value 145.918786
## stopped after 100 iterations
```

```
ex3_fit_time<-Sys.time() - ex3_fit_time



train_pred_ex3 <- predict(ex3.fit, newdata=dt_ex3_train, type="raw")
test_pred_ex3 <- predict(ex3.fit, newdata=dt_ex3_test, type="raw")
```

4. Compare these three NN model at least in two characteristics: the performance and CPU time.

```
taulahog10<-(table(test_pred_10, max.col(dt_HOG_test[,names(dt_HOG_test)%in%resp_var_HOG])));
cmhog10<-confusionMatrix(taulahog10);cmhog10
```

```
## Confusion Matrix and Statistics
##
```

```
##
## test_pred_10  1  2  3  4  5
##            1 18  3  1  2  0
##            2  3 18  0  7  0
##            3  2  0 20  2  1
##            4  3  3  5 14  0
##            5  0  1  0  0 24
##
## Overall Statistics
##
##                Accuracy : 0.7401575
##                  95% CI : (0.6548765, 0.813901)
##     No Information Rate : 0.2047244
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.6752422
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1  Class: 2  Class: 3  Class: 4  Class: 5
## Sensitivity          0.6923077 0.7200000 0.7692308 0.5600000 0.9600000
## Specificity          0.9405941 0.9019608 0.9504950 0.8921569 0.9901961
## Pos Pred Value       0.7500000 0.6428571 0.8000000 0.5600000 0.9600000
## Neg Pred Value       0.9223301 0.9292929 0.9411765 0.8921569 0.9901961
## Prevalence           0.2047244 0.1968504 0.2047244 0.1968504 0.1968504
## Detection Rate       0.1417323 0.1417323 0.1574803 0.1102362 0.1889764
## Detection Prevalence 0.1889764 0.2204724 0.1968504 0.1968504 0.1968504
## Balanced Accuracy    0.8164509 0.8109804 0.8598629 0.7260784 0.9750980
```

```
taulahog50<-(table(test_pred_50, max.col(dt_HOG_test[,names(dt_HOG_test)%in%resp_var_HOG])));
cmhog50<-confusionMatrix(taulahog50);cmhog50
```

```
## Confusion Matrix and Statistics
##
##
## test_pred_50  1  2  3  4  5
##            1 19  2  1  1  0
##            2  2 19  0  8  0
##            3  2  2 19  2  1
##            4  3  1  6 14  0
##            5  0  1  0  0 24
##
## Overall Statistics
##
##                Accuracy : 0.7480315
##                  95% CI : (0.6633077, 0.8208324)
##     No Information Rate : 0.2047244
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.6850833
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

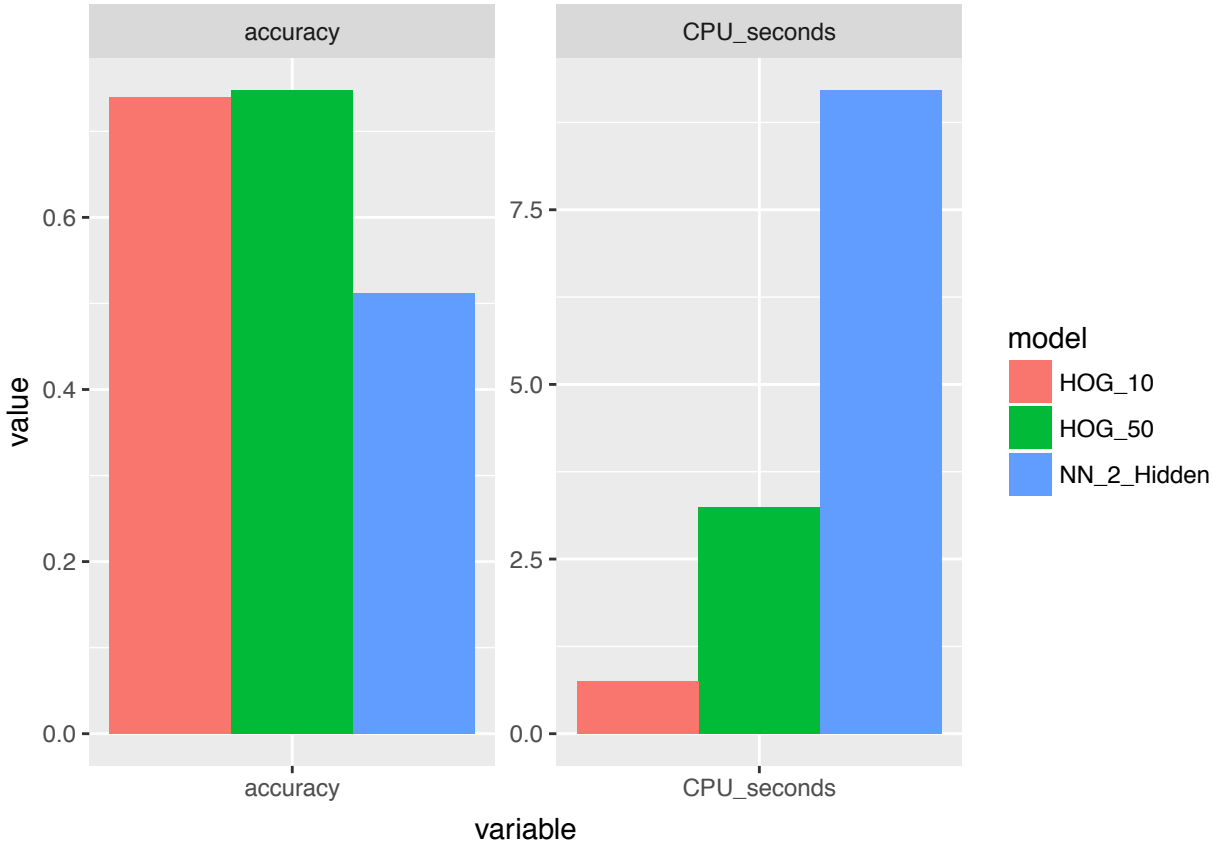
```
##                       Class: 1  Class: 2  Class: 3  Class: 4  Class: 5
## Sensitivity          0.7307692 0.7600000 0.7307692 0.5600000 0.9600000
## Specificity          0.9603960 0.9019608 0.9306931 0.9019608 0.9901961
## Pos Pred Value       0.8260870 0.6551724 0.7307692 0.5833333 0.9600000
## Neg Pred Value       0.9326923 0.9387755 0.9306931 0.8932039 0.9901961
## Prevalence           0.2047244 0.1968504 0.2047244 0.1968504 0.1968504
## Detection Rate       0.1496063 0.1496063 0.1496063 0.1102362 0.1889764
## Detection Prevalence 0.1811024 0.2283465 0.2047244 0.1889764 0.1968504
## Balanced Accuracy    0.8455826 0.8309804 0.8307312 0.7309804 0.9750980
```

```
taulaex3<-(table(test_pred_ex3, dt_ex3_test[,'label']));
cmex3<-confusionMatrix(taulaex3); cmex3
```

```
## Confusion Matrix and Statistics
##
##
## test_pred_ex3 helicopter kangaroo menorah starfish trilobite
##    helicopter         13        3       3        7         1
##    kangaroo            5       11       9        5         0
##    menorah             2        5      11        1         0
##    starfish            6        5       3       12         0
##    trilobite           0        1       0        0        24
##
## Overall Statistics
##
##                Accuracy : 0.5590551
##                  95% CI : (0.4682592, 0.6470337)
##     No Information Rate : 0.2047244
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.4490239
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: helicopter Class: kangaroo Class: menorah
## Sensitivity                  0.5000000      0.44000000     0.42307692
## Specificity                  0.8613861      0.81372549     0.92079208
## Pos Pred Value               0.4814815      0.36666667     0.57894737
## Neg Pred Value               0.8700000      0.85567010     0.86111111
## Prevalence                   0.2047244      0.19685039     0.20472441
## Detection Rate               0.1023622      0.08661417     0.08661417
## Detection Prevalence         0.2125984      0.23622047     0.14960630
## Balanced Accuracy            0.6806931      0.62686275     0.67193450
##                      Class: starfish Class: trilobite
## Sensitivity               0.48000000        0.9600000
## Specificity               0.86274510        0.9901961
## Pos Pred Value            0.46153846        0.9600000
## Neg Pred Value            0.87128713        0.9901961
## Prevalence                0.19685039        0.1968504
## Detection Rate            0.09448819        0.1889764
## Detection Prevalence      0.20472441        0.1968504
## Balanced Accuracy         0.67137255        0.9750980
```

```
data.frame(accuracy=c(cmhog10$overall[1],cmhog50$overall[1],cmex3$overall[1]),
           CPU_seconds=c(HOG_10_CPU_time,HOG_50_CPU_time, ex3_fit_time),
           model=c("HOG_10","HOG_50","NN_2_Hidden")) %>%
  melt(id.var="model") %>%
  ggplot(aes(x=variable,
             y=value,
             fill=model)) +
  geom_bar(stat='identity',
           position="dodge") +
  facet_wrap(~variable,scales="free")
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```



We believe that the three models are not comparable. The HOG ones are, but comparing them to the NN with 2 hidden layers makes less sense, This is due to: - The format of the images is not the same. HOG is a vector of 81 elements, whereas the neural network has a longer length: 1024. Therefore, HOG_50 takes longer than HOG_10 as it is more complex; but comparing it with the other alternative makes less sense. The Neural Network is the one that takes longer to compute. On the other hand, Neural Network with caret implements a 5 fold CV, whereas the raw implementation of neuralnet doesnt.

PERFORMANCE: In terms of performance, HOG_10 has the best one.

CNN architectures:

Categories: airplanes (800), Motorbikes (798) and Faces (435) are large categories. For each category reshape grayscale images to 32x32. Split the dataset in two halves (training and test). From the training part, fit a CNN according with the model

```
categories_ex5 <- c("airplanes","Motorbikes","Faces")

obtain_images_to_keras <- function(x,vector_categories){

  #gets the list of the files in a given directory first.
  path_imgs <- list.files(paste0(images_dir,vector_categories[x]),
                  pattern = ".jpg",
                  full.names = TRUE)
  #creates an auxiliary matrix where we will place the results.
  #every row is a image. it first contains the label, and then the 1024 vector
  dt_img<-list()
  #we iterate on all the images
  for(i in 1:length(path_imgs)) {
    img <- image_to_grey(path_imgs[i]) %>%
      img2arr()
    dt_img [[i]]<- img
    }
  dt_img <- array(unlist(dt_img), dim = c(length(path_imgs), 32, 32))
  return(dt_img)
}


airplanes.array<-obtain_images_to_keras(1,categories_ex5)
dim(airplanes.array)
motorbikes.array<-obtain_images_to_keras(2,categories_ex5)
dim(motorbikes.array)
faces.array<-obtain_images_to_keras(3,categories_ex5)
dim(faces.array)


dt_ex5 <- abind(obtain_images_to_keras(1,categories_ex5),
            obtain_images_to_keras(2,categories_ex5),along=1)
dt_ex5 <- abind(dt_ex5,obtain_images_to_keras(3,categories_ex5),along=1)


labels_ex5 <- c(rep(categories_ex5[1],number_images[categories==categories_ex5[1]]),
           rep(categories_ex5[2],number_images[categories==categories_ex5[2]]),
           rep(categories_ex5[3],number_images[categories==categories_ex5[3]])) %>%
  as.factor() %>%
  as.numeric

index <- sample(1:dim(dt_ex5)[1],(2/3)*dim(dt_ex5)[1])

x_train <- dt_ex5[index,,]
x_test <- dt_ex5[-index,,]

x_train <- array_reshape(x_train, c(nrow(x_train), 32, 32 ,1))
x_test <- array_reshape(x_test, c(nrow(x_test), 32, 32 ,1))
```

```
y_train<- array(labels_ex5[index])
y_test<- array(labels_ex5[-index])
y_train <- to_categorical(y_train-1, 3)
y_test <- to_categorical(y_test-1, 3)

input_shape<-c(32, 32, 1)

#checking the dimensions
dim(x_train)
```

5. Convolution layer with: . filters = 20, kernel_size = c(3,3), activation = 'relu', . Pooling layer with: pool_size = c(2, 2) . Fully connected layer with: units = 128, activation = 'relu' . Output layer with: units = num_classes, activation = 'softmax' . Assess the performance of the CNN predicting the categories of test images and obtain the confusion matrix.

```
num_classes<-3
start5time<-Sys.time()
##Model Convolutional Neural Network. documentation from --> https://keras.io/
###En primer lloc cal instalar Anaconda. En segon lloc executes: library(keras) , install_keras(). Un c
input.shape<-c(32,32,1)
time<-Sys.time()
model5 <-keras_model_sequential()%>%
layer_conv_2d(filters=20, kernel_size=c(3,3),activation='relu', input_shape = input.shape)%>%
layer_max_pooling_2d(pool_size=c(2,2))%>%
layer_flatten()%>%
layer_dense(units=128, activation='relu')%>%
layer_dense(units=num_classes, activation='softmax')

#opt<-optimizer_adam( lr= 0.0001 , decay = 1e-6 )
model5 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = 'accuracy'
)

summary(model5)

model5cnnet <- model5 %>% fit(
  x_train, y_train,
  epochs = 20, batch_size = 128,
  validation_split = 0.2
)
end5tyme<-Sys.time()
plotmodel5cnnet<-plot(model5cnnet);plotmodel5cnnet
```

```
model5cnnettest <- predict(model5, x_test)
```

6. From the training part, fit a CNN according with a model proposed by yourself. . Describe the model. . Assess the performance of the CNN predicting the categories of test images and obtain the confusion matrix . filters = 30, kernel_size = c(3,3), activation = 'tanh', . Pooling layer with: pool_size = c(3, 3) . Fully connected layer with: units = 128, activation = 'tanh' . Output layer with: units = num_classes, activation = 'sigmoid' . Assess the performance of the CNN predicting the categories of test images and obtain the confusion matrix. **

```
start6tyme<-Sys.time()
input.shape<-c(32,32,1)
time<-Sys.time()
model6 <-keras_model_sequential()%>%
layer_conv_2d(filters=30, kernel_size=c(3,3),activation='tanh', input_shape = input.shape)%>%
layer_max_pooling_2d(pool_size=c(3,3))%>%
layer_flatten()%>%
layer_dense(units=128, activation='tanh')%>%
layer_dense(units=3, activation='sigmoid')

#opt<-optimizer_adam( lr= 0.0001 , decay = 1e-6 )
model6 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = 'accuracy'
)

summary(model6)

model6cnnet <- model6 %>% fit(
  x_train, y_train,
  epochs = 12, batch_size = 128,
  validation_split = 0.2
)
end6tyme<-Sys.time()
plotmodel6cnnet<-plot(model6cnnet);plotmodel6cnnet
```

```
model6cnnettest <- predict(model6, x_test)
```

7. Compare these two CNN model at least in two characteristics: the performance and CPU time.

```
timemodel5cnnet<-end5tyme - start5time ;timemodel5cnnet
timemodel6cnnet<-end6tyme - start6tyme ;timemodel6cnnet
```

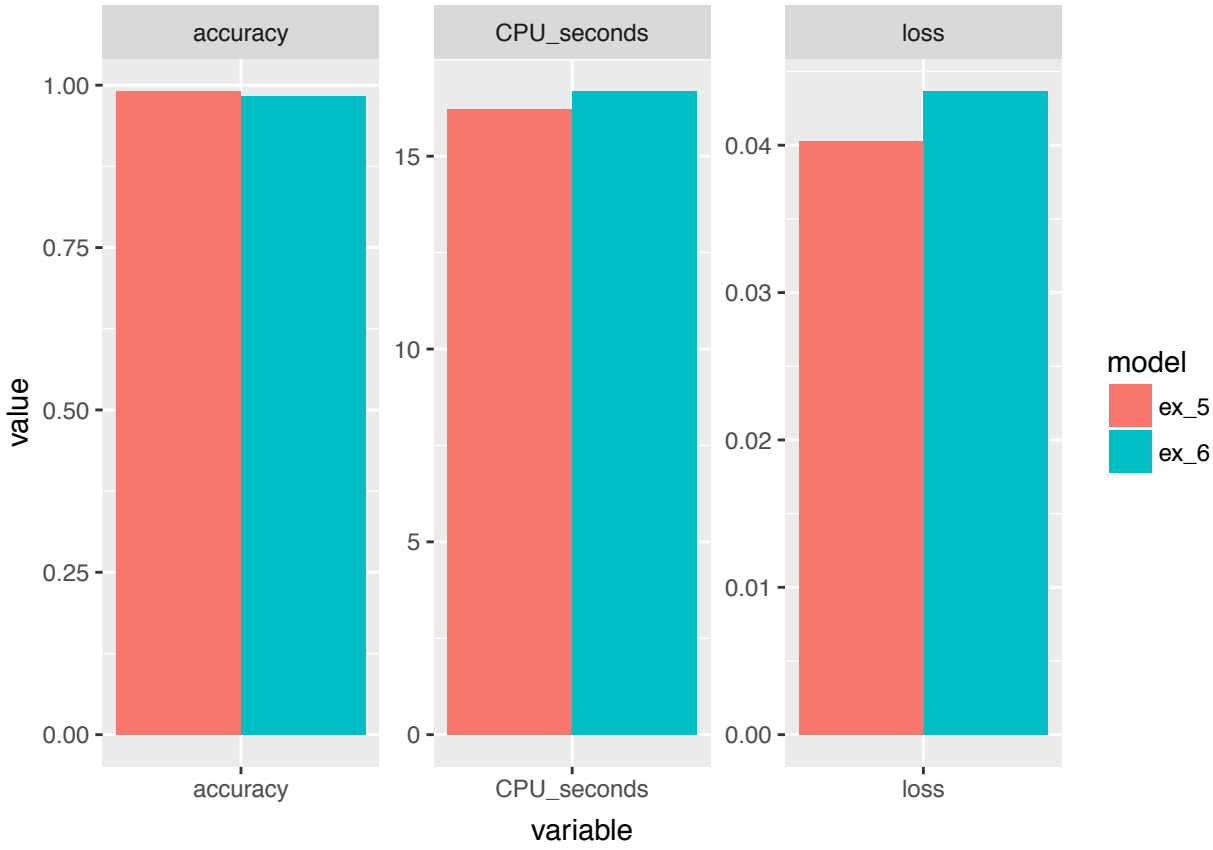
```
loss5<-summary(model5cnnet$metrics$loss);loss5
loss6<-summary(model6cnnet$metrics$loss);loss6
```

```
accuracy5<-summary(model5cnnet$metrics$acc);accuracy5
accuracy6<-summary(model6cnnet$metrics$acc);accuracy6
```

```
imgs_to_load<-list.files("objects_to_load/objectesR")
for (i in 1:length(imgs_to_load)){
  load(paste0("objects_to_load/objectesR/",imgs_to_load[i]))
}

data.frame(accuracy=c(accuracy5,accuracy6),
           CPU_seconds=c(timemodel5cnnet,timemodel6cnnet),
           loss=c(loss5,loss6),
           model=c("ex_5","ex_6")) %>%
  melt(id.var="model") %>%
  ggplot(aes(x=variable,
             y=value,
             fill=model)) +
  geom_bar(stat='identity',
```

```
          position="dodge") +
  facet_wrap(~variable,scales="free")
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```



The best CNN model is the model5cnnet. Although its CPU time is sligthly higher than in model6cnnet, its performance is better. It also has a lower loss.

---

8. Compare the NN (in 3) with the CNN (in 5) in the analysis of airplanes, Motorbikes and Faces categories.

```
categories_ex5 <- c("airplanes","Motorbikes","Faces")
dt_ex8 <- rbind(obtain_images(1,categories_ex5),
            obtain_images(2,categories_ex5),
            obtain_images(3,categories_ex5))
```

```
trainIndex <- createDataPartition(dt_ex8$label, p=.7, list=F)
dt_ex8_train <- dt_ex8[trainIndex, ]
dt_ex8_test <- dt_ex8[-trainIndex, ]

xnam <- names(dt_ex8_train[-1])
resp_var <- paste0(levels(dt_ex8_train$label),collapse=" + ")
(fmla <- as.formula(paste(resp_var, "~", paste(xnam, collapse= "+"))))
```

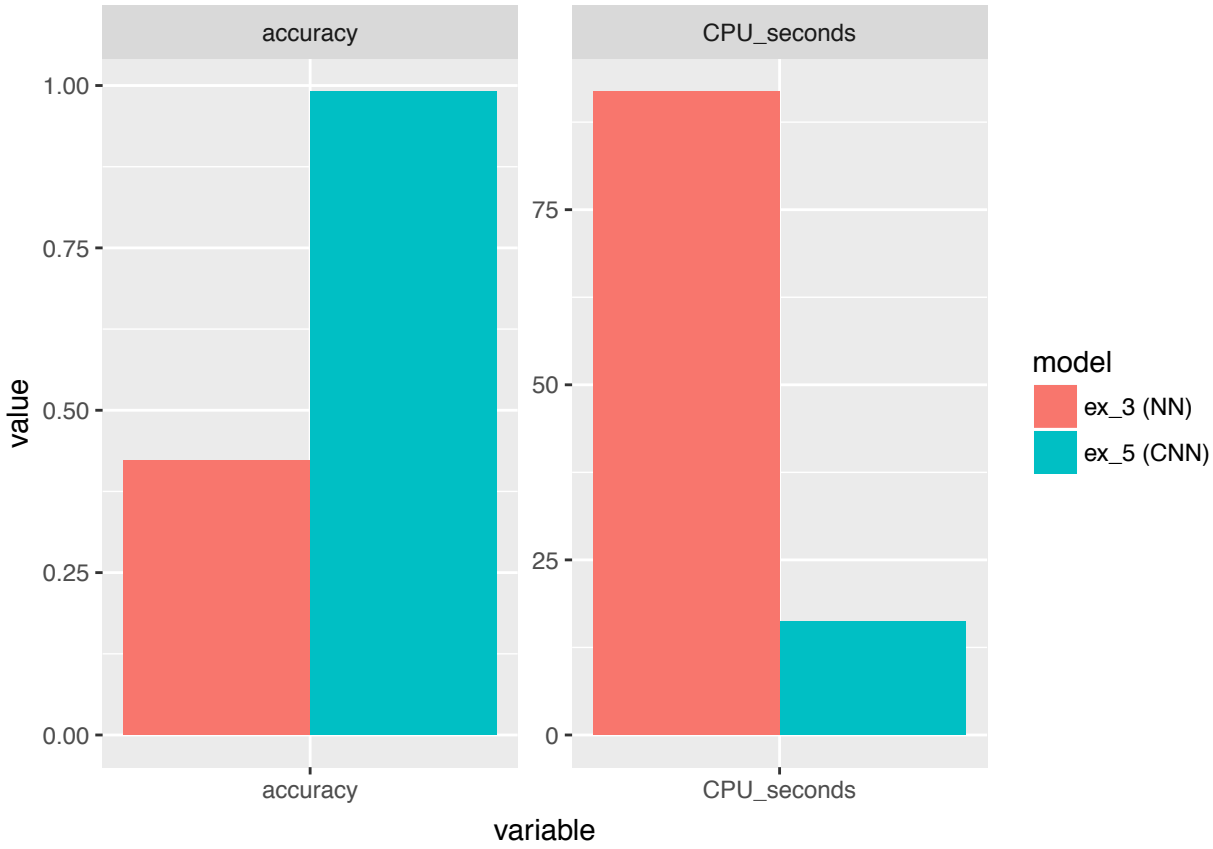
```
my.grid <- expand.grid(size=seq(from=1,to=5,by=3),
                       decay=seq(from=0.1,to=0.9,by=0.3))
ex_8_grid_search_time<-Sys.time()
ex8.fit <- train(label ~ .,
                 data = dt_ex8_train,
                 method = "nnet",
                 tuneGrid = my.grid,
                 MaxNWts=6036,
                 trControl = trainControl(method = "cv",
                                          verboseIter = FALSE,
                                          number = 2,
                                          returnData = FALSE),
                 verbose=FALSE)
ex_8_grid_search_time<-Sys.time() - ex_8_grid_search_time

save(ex_8_grid_search_time, file="objects_to_load/objectesR/ex_8_grid_search_time.RData")

#for the optimized parameters
my.grid <- expand.grid(size=ex8.fit$bestTune$size,
                       decay=ex8.fit$bestTune$decay)
ex_8_fit_time <- Sys.time()
ex8.fit <- train(label ~ .,
                 data = dt_ex8_train,
                 method = "nnet",
                 tuneGrid = my.grid,
                 MaxNWts=6036,
                 trControl = trainControl(method = "cv",
                                          verboseIter = FALSE,
                                          number = 2,
                                          returnData = FALSE),
                 verbose=FALSE)
ex_8_fit_time <- Sys.time() - ex_8_fit_time

# Plot model and predict
#plotnet(ex8.fit)
train_pred_ex8 <- predict(ex8.fit, newdata=dt_ex8_train[,-1], type="raw")
test_pred_ex8 <- predict(ex8.fit, newdata=dt_ex8_test[,-1], type="raw")
```

```
imgs_to_load<-list.files("objects_to_load/objectesR")
for (i in 1:length(imgs_to_load)){
  load(paste0("objects_to_load/objectesR/",imgs_to_load[i]))
}

taulaex8<-(table(test_pred_ex8, dt_ex8_test[,1]));
cmex8<-confusionMatrix(taulaex8); sum(diag(cmex8$table))/sum(cmex8$table)

save(ex_8_fit_time, file="objects_to_load/objectesR/ex_8_fit_time.RData")
save(cmex8, file="objects_to_load/objectesR/cmex8.RData")
```

Doing the grid search is extremely costly for the NN in the nnet integration of caret. It took 4.1993806163 to do the grid.search, and after that 1.5317859491 to just fit the model for the optimized parameters. It was also limited using the parameter MaxNWts, otherwise the computation was taking really long.

```
imgs_to_load<-list.files("objects_to_load/objectesR")
for (i in 1:length(imgs_to_load)){
  load(paste0("objects_to_load/objectesR/",imgs_to_load[i]))
}


data.frame(accuracy=c(accuracy5,sum(diag(cmex8$table))/sum(cmex8$table)),
           CPU_seconds=c(timemodel5cnnet,ex_8_fit_time),
           model=c("ex_5 (CNN)","ex_3 (NN)")) %>%
  melt(id.var="model") %>%
  ggplot(aes(x=variable,
             y=value,
             fill=model)) +
  geom_bar(stat='identity',
           position="dodge") +
  facet_wrap(~variable,scales="free")
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```



Both performance wise (accuracy) and in terms of CPU time, the CNN is way better than the NN.