PCA-and-tuning-implementation-in-keras-DNN-R

Gyan Shashwat

07/03/2020

Prediction model for daily and sports activities recognition

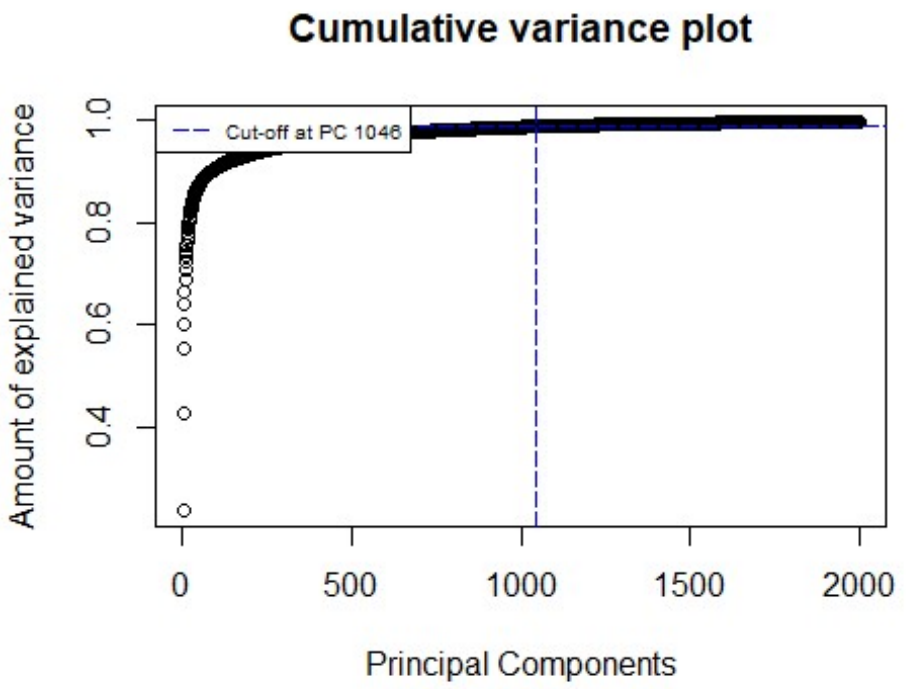
```
# load packages and data0
library(keras)
load("data_activity_recognition.RData")
# we convert these into 28x28 vectors
x_train <- array_reshape(x_train, c(nrow(x_train), 125*45))
x_test <- array_reshape(x_test, c(nrow(x_test), 125*45))
range_norm <- function(x, a = 0, b = 1) {
( (x - min(x)) / (max(x) - min(x)) )*(b - a) + a
}
x_train <- apply(x_train, 2,range_norm)
x_test <- apply(x_test, 2,range_norm)
# split the test data in two halves: one for validation
# and the other for actual testing
#val <- sample(1:nrow(x_test), 750)
# there are 10000 images in x_test
#test <- setdiff(1:nrow(x_test), val)
#x_val <- x_test[val,]
#x_newtest <- x_test[test,]
# need tese later
#N <- nrow(x_train)
#V <- ncol(x_train)
```

Including Plots

You can also embed plots, for example:

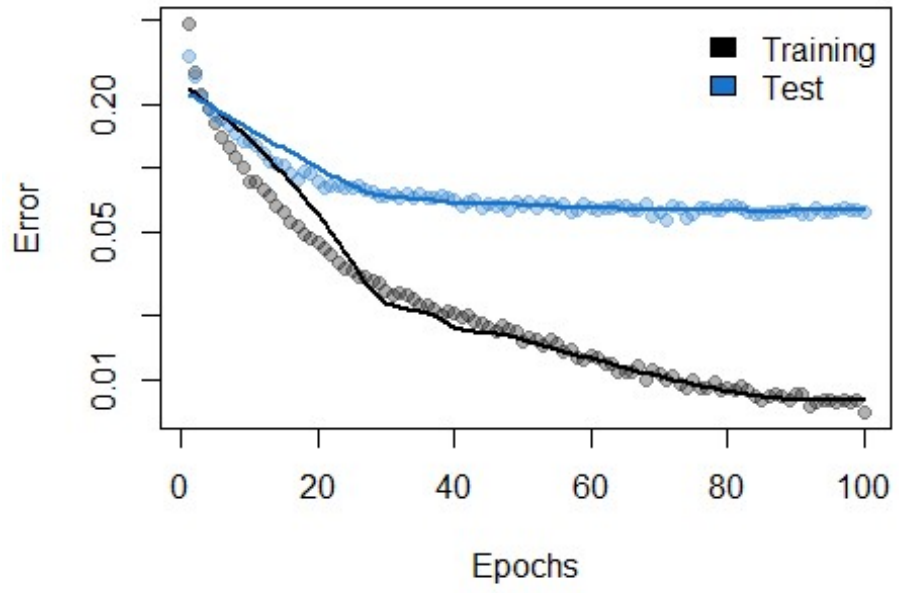
```
library(MASS)
pca <- prcomp(x_train)

save(pca, file="pca.RData")

load("pca.RData")
prop <- cumsum(pca$sdev^2)/sum(pca$sdev^2)
Q <- length( prop[prop < 0.99] )
xz_train <- pca$x[,1:Q]
# extract first Q principal components
plot(prop[0:2000], xlab = "Principal Components", ylab = "Amount of explained
variance", main = "Cumulative variance plot", col="black")
abline(v = 1046, col="blue", lty=5)
abline(h = 0.99, col="blue", lty=5)
```

```
legend("topleft", legend=c("Cut-off at PC 1046"), col=c("blue"), lty=5,
cex=0.6)
```



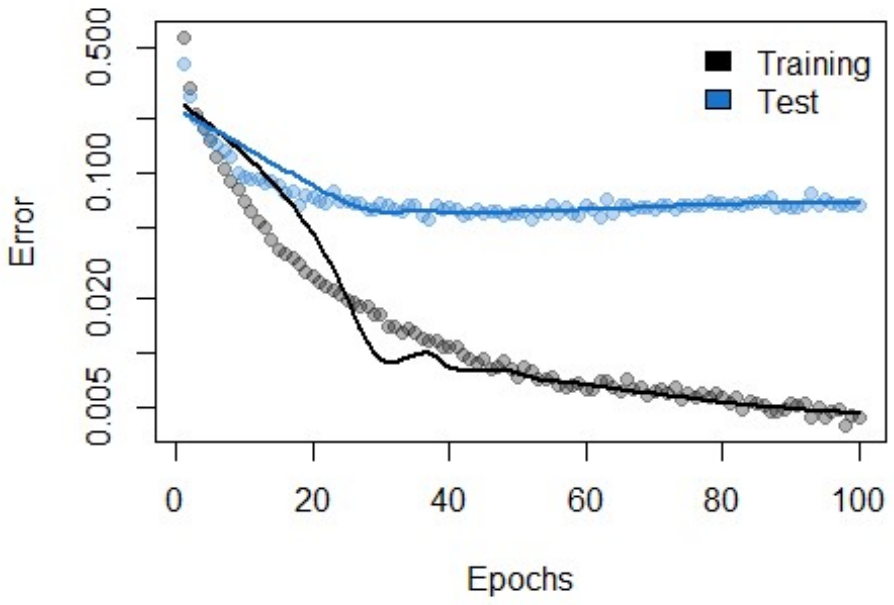

```
# map original test data points on to the learned subspace
xz_test <- predict(pca, x_test)[,1:Q]

#install_keras()
y_train<-factor(y_train)
y_train <- to_categorical(as.numeric(y_train)-1, num_classes = 19)
y_test<-factor(y_test)
y_test<-to_categorical(as.numeric(y_test)-1, num_classes = 19)

V <- ncol(xz_train)
N<-nrow(xz_train)
# 256
singlemodel <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu", input_shape = V) %>%
  layer_dense(units = 19, activation = "softmax") %>%
  compile(
    loss = "categorical_crossentropy", metrics = "accuracy",
    optimizer = optimizer_sgd(),
  )
# one-hot encoding of target variable
fit <- singlemodel %>% fit(
  x = xz_train, y = y_train,
  validation_data = list(xz_test, y_test),
  epochs = 100,
```

```
  verbose = 0
)

# to add a smooth line to points
smooth_line <- function(y) {
  x <- 1:length(y)
  out <- predict( loess(y ~ x) )
  return(out)
}
# some colors will be used later
cols <- c("black", "dodgerblue3", "gray50", "deepskyblue2")
# check performance ---> error
outsingle <- 1 - cbind(fit$metrics$accuracy,
                       fit$metrics$val_accuracy)
matplot(outsingle, pch = 19, ylab = "Error", xlab = "Epochs",
        col = adjustcolor(cols[1:2], 0.3),
        log = "y")
# on log scale to visualize better differences
matlines(apply(outsingle, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("topright", legend = c("Training", "Test"),
       fill = cols[1:2], bty = "n")
```




```
singlemodel %>% evaluate(xz_test, y_test, verbose = 0)

## $loss
## [1] 0.1606897
##
```

```
## $accuracy
## [1] 0.9381579

V <- ncol(xz_train)
N<-nrow(xz_train)
# 256
modeldouble <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu", input_shape = V) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dense(units = 19, activation = "softmax") %>%
  compile(
    loss = "categorical_crossentropy", metrics = "accuracy",
    optimizer = optimizer_sgd(),
  )

# one-hot encoding of target variable
fit <- modeldouble %>% fit(
  x = xz_train, y = y_train,
  validation_data = list(xz_test, y_test),
  epochs = 100,
  verbose = 0
)

# to add a smooth line to points
smooth_line <- function(y) {
  x <- 1:length(y)
  out <- predict( loess(y ~ x) )
  return(out)
}
# some colors will be used later
cols <- c("black", "dodgerblue3", "gray50", "deepskyblue2")
# check performance ---> error
outdouble <- 1 - cbind(fit$metrics$accuracy,
                 fit$metrics$val_accuracy)
matplot(outdouble, pch = 19, ylab = "Error", xlab = "Epochs",
        col = adjustcolor(cols[1:2], 0.3),
        log = "y")
# on log scale to visualize better differences
matlines(apply(outdouble, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("topright", legend = c("Training", "Test"),
       fill = cols[1:2], bty = "n")
```

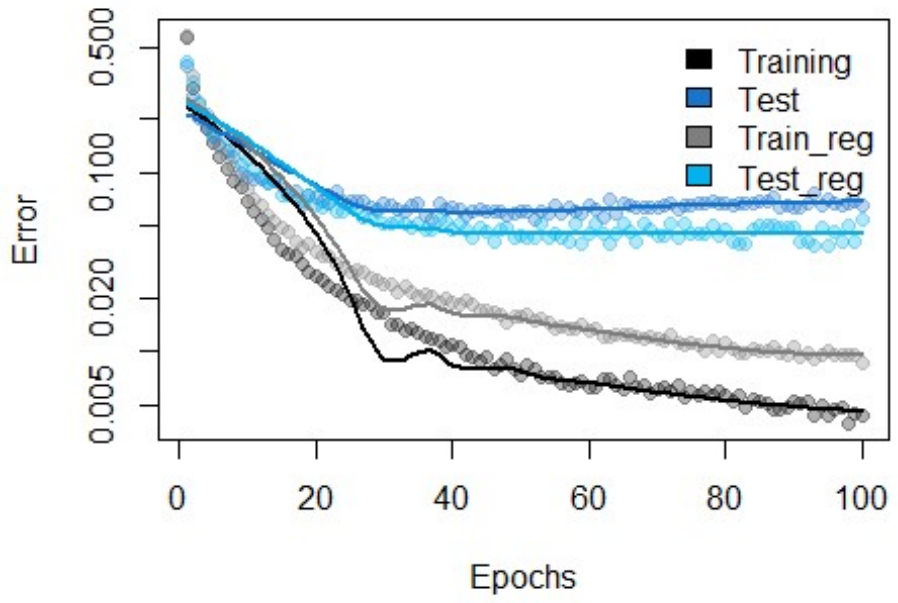



```
modeldouble %>% evaluate(xz_test, y_test, verbose = 0)
```

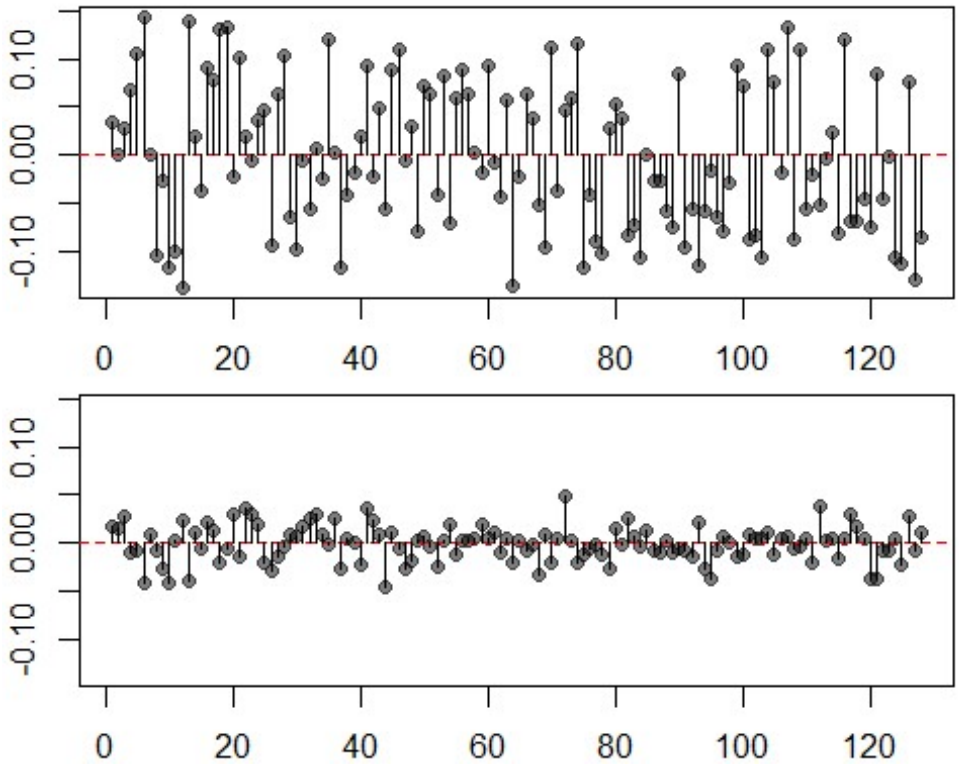
```
## $loss
## [1] 0.1835015
##
## $accuracy
## [1] 0.9342105
```

```
#keras sequential model for multilayer neural network with one extra layer
model_regularised <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu", input_shape =V,
kernel_regularizer = regularizer_l2(l = 0.004))%>% #First hidden layer
considering 256 units  and relu as activation function and input shape as
value of V, kernel_regularizer as regularizer_l2() for wieght decay
regularisation , l is the hyperparameter, here value is 0.004 (selected via
hit and trail)

  layer_dense(units = 128, activation = "relu",
              kernel_regularizer = regularizer_l2(l = 0.004))%>%
  #First hidden layer considering 128 units  and relu as activation function
, kernel_regularizer as regularizer_l2() for wieght decay regularisation ,
and l is the hyperparameter, here value is 0.004 (selected via hit and trail)

  layer_dense(units = 19, activation = "softmax") %>% #Output layer
considering 10 output units and softmax as activation function
   #compling the above model by taking cross entropy as error function,
accuracy as performance measure and stochastic gradient descent for
```

```
optimization.
  compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_sgd(),
    metrics = "accuracy"
  )
# count parameters of regularised model
# Model fit for regularised multiple neural netwrok with 2 hidden layers.
# train and evaluate on test data at each epoch
fit_reg <- model_regularised %>% fit(
  x = xz_train, y = y_train,#traning data for the model
  validation_data = list(xz_test, y_test),#testing data for validation
  epochs = 100,#no of epoch
  verbose = 0
)

out_reg <- 1 - cbind(fit$metrics$accuracy,
                     fit$metrics$val_accuracy,
                     fit_reg$metrics$accuracy,
                     fit_reg$metrics$val_accuracy)#data with performance
parameters for regularised model
# check performance
matplot(out_reg, pch = 19, ylab = "Error", xlab = "Epochs",
        col = adjustcolor(cols, 0.3),
        log = "y")#matrix plot of performance at each epoch
matlines(apply(out_reg, 2, smooth_line), lty = 1, col = cols, lwd = 2)#matrix
lines connecting each points
legend("topright", legend = c("Training", "Test", "Train_reg", "Test_reg"),
       fill = cols, bty = "n")#legend for the plot.
```



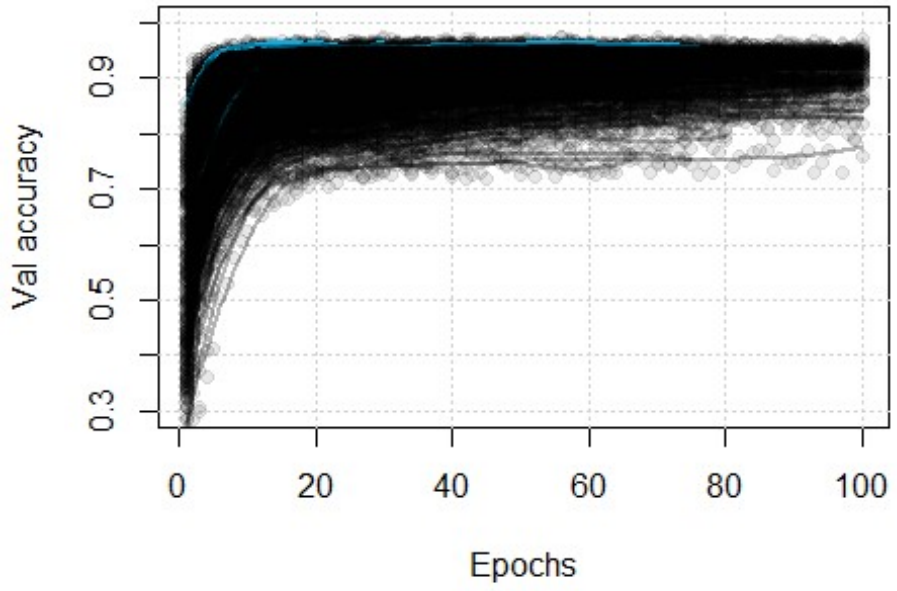

```
apply(out_reg, 2, min)#minimun value of each error colum for training and
testing datset for both regularised and non regularised models.

## [1] 0.003947377 0.054605246 0.008684218 0.038815796

# get all weights
w_all <- get_weights(modeldouble) #all weights for 2 hidden layer model
w_all_reg <- get_weights(model_regularised) #all weights for regularised 2
hidden layer model
# weights of first hidden layer
# one input --> 64 units
w <- w_all[[3]][1,]
w_reg <- w_all_reg[[3]][1,]
# compare visually the magnitudes
par(mfrow = c(2,1), mar = c(2,2,0.5,0.5))#non regularised plot
r <- range(w)
n <- length(w)
plot(w, ylim = r, pch = 19, col = adjustcolor(1, 0.5))#adding lines in non
regularised plot
abline(h = 0, lty = 2, col = "red")
segments(1:n, 0, 1:n, w)
#
plot(w_reg, ylim = r, pch = 19, col = adjustcolor(1, 0.5))# regularised plot
abline(h = 0, lty = 2, col = "red")#adding lines in regularised plot
segments(1:n, 0, 1:n, w_reg)
```




```r
V<-ncol(xz_train)
N<-nrow(xz_train)
library(tfruns)
# split the test data in two halves: one for validation
# and the other for actual testing
set.seed(19200276)
# there are 2007 images in x_test
val <- sample(1:nrow(xz_test),760) #smaple rows for validation data
test <- setdiff(1:nrow(xz_test), val) # smaple rows for test data
x_val <- xz_test[val,] # predictor variable for validation data
y_val <- y_test[val,] # response variable for validation data
xtun_test <- xz_test[test,] #predictor variable for test data
ytun_test <- y_test[test,] #response variable for test data
# flags grid of nodes for 3 hidden layer and dropout
size1_set=c(256,128,64)
size2_set=c(256,128,64)
size3_set=c(256,128,64)
dropout_set=c(0,0.3,0.5,0.6)
lambda_set <- c(0, exp( seq(-6, -4, length = 9) ))
#running the model
runs <- tuning_run("tuning.R",
                   runs_dir = "tuning_pca",
                   flags = list(
                     dropout = dropout_set,
                     unit1 = size1_set,
                     unit2 = size2_set,
                     unit3 = size3_set,
```

```
                    lambda=lambda_set
                  ),sample = 0.3)

library(jsonlite) #importing jsonlite pakage
library(doParallel)

## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel

library(tfruns)
read_metrics <- function(path, files = NULL)
  # 'path' is where the runs are --> e.g. "path/to/runs"
{
  path <- paste0(path, "/")
  if ( is.null(files) ) files <- list.files(path)
  n <- length(files)
  out <- vector("list", n)
  for ( i in 1:n ) {
    dir <- paste0(path, files[i], "/tfruns.d/")
    out[[i]] <- jsonlite::fromJSON(paste0(dir, "metrics.json"))
    out[[i]]$flags <- jsonlite::fromJSON(paste0(dir, "flags.json"))
    out[[i]]$evaluation <- jsonlite::fromJSON(paste0(dir, "evaluation.json"))
  }
  return(out)
}
plot_learning_curve <- function(x, ylab = NULL, cols = NULL, top = 3, span =
0.4, ...)
{
  # to add a smooth line to points
  smooth_line <- function(y) {
    x <- 1:length(y)
    out <- predict( loess(y ~ x, span = span) )
    return(out)
  }
  matplot(x, ylab = ylab, xlab = "Epochs", type = "n", ...)
  grid()
  matplot(x, pch = 19, col = adjustcolor(cols, 0.3), add = TRUE)
  tmp <- apply(x, 2, smooth_line)
  tmp <- sapply( tmp, "length<-", max(lengths(tmp)) )
  set <- order(apply(tmp, 2, max, na.rm = TRUE), decreasing = TRUE)[1:top]
  cl <- rep(cols, ncol(tmp))
  cl[set] <- "deepskyblue2"
  matlines(tmp, lty = 1, col = cl, lwd = 2)
}



out <- read_metrics("tuning_pca")
# extract validation accuracy and plot learning curve
```

```
acc <- sapply(out, "[[", "val_accuracy")
plot_learning_curve(acc, col = adjustcolor("black", 0.3), ylim = c(0.3, 1),
                    ylab = "Val accuracy", top = 3)
```

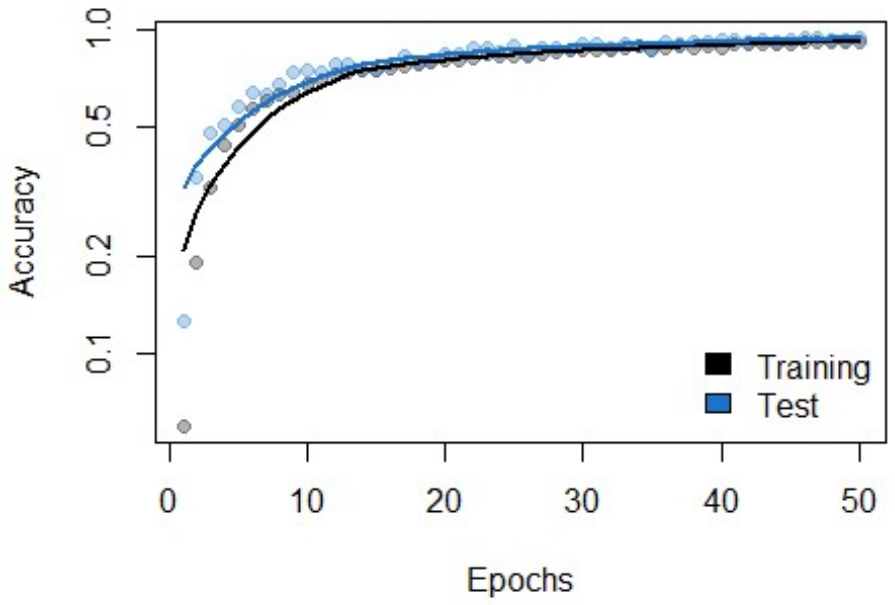



```
# all flag value result object
res <- ls_runs(metric_val_accuracy > 0.92,
               runs_dir = "tuning_pca", order = metric_val_accuracy)

res <- res[,c(2,4,8:14)]
res[1:10,]

## Data frame: 10 x 9
##    metric_val_accuracy eval_accuracy flag_dropout flag_unit1 flag_unit2
## 1               0.9697        0.9526          0.3        128         64
## 2               0.9645        0.9513          0.3        256        128
## 3               0.9632        0.9539          0.0         64         64
## 4               0.9618        0.9618          0.0        128        128
## 5               0.9618        0.9658          0.0        256        128
## 6               0.9605        0.9513          0.0        256        128
## 7               0.9592        0.9487          0.0         64         64
## 8               0.9592        0.9566          0.3        256        128
## 9               0.9579        0.9553          0.0        128        128
## 10              0.9579        0.9592          0.0        256        128
##    flag_unit3 flag_lambda samples batch_size
## 1         256      0.0067    7600        152
## 2         256      0.0052    7600        152
## 3          64      0.0052    7600        152
```

```
## 4          64      0.0052    7600         152
## 5         128      0.0052    7600         152
## 6          64      0.0087    7600         152
## 7         256      0.0087    7600         152
## 8         256      0.0067    7600         152
## 9          64      0.0143    7600         152
## 10         64      0.0143    7600         152
```

```
load("data_activity_recognition.RData")
x_train <- array_reshape(x_train, c(nrow(x_train), 125, 45, 1))
x_test <- array_reshape(x_test, c(nrow(x_test), 125, 45, 1))
y_train<-factor(y_train)
y_train <- to_categorical(as.numeric(y_train)-1, num_classes = 19)
y_test<-factor(y_test)
y_test<-to_categorical(as.numeric(y_test)-1, num_classes = 19)
x_train<-x_train/255
x_test<-x_test/255
cNNmodel <- keras_model_sequential() %>%
#
# convolutional layers
layer_conv_2d(filters = 32, kernel_size = c(2,2), activation = "relu",
input_shape = c(125, 45, 1)) %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
#
# fully connected layers
layer_flatten() %>%
layer_dense(units = 64, activation = "relu", kernel_regularizer =
regularizer_l2(0.1)) %>%
layer_dropout(0.1) %>%
layer_dense(units = ncol(y_train), activation = "softmax") %>%
#
# compile
compile(
loss = "categorical_crossentropy",
metrics = "accuracy",
optimizer = optimizer_adam()
)
# model training
# NOTE : this will require some time
fit <- cNNmodel %>% fit(
x = x_train, y = y_train,
validation_data = list(x_test, y_test),
epochs = 50,
batch_size = 256,
verbose = 0
# roughly 0.5% of training observations
```

```
)

# to add a smooth line to points
smooth_line <- function(y) {
x <- 1:length(y)
out <- predict( loess(y ~ x) )
return(out)
}
# check performance
cols <- c("black", "dodgerblue3")
covout <- cbind(fit$metrics$accuracy,
fit$metrics$val_accuracy)
matplot(covout, pch = 19, ylab = "Accuracy", xlab = "Epochs",
col = adjustcolor(cols[1:2], 0.3),
log = "y")
matlines(apply(covout, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("bottomright", legend = c("Training", "Test"),
fill = cols[1:2], bty = "n")
```




```
cNNmodel %>% evaluate(x_test, y_test, verbose = 0)

## $loss
## [1] 0.3440312
##
## $accuracy
## [1] 0.9427631
```

```r
class_labels <- c("asc_stairs", "basketball", "cross_trainer",
"cycling_horiz", "cycling_vert","desc_stairs", "jumping", "lying_back",
"lying_side",
"moving_elevator","rowing","running_treadmill","sitting","stand_elevator","st
anding","stepper","walking","walking_tread","walking_tread_incl")
# look at classification table
class_y <- class_labels[max.col(y_test)]
class_hat <- class_labels[ cNNmodel %>% predict_classes(x_test) + 1 ]
tab <- table(class_y, class_hat)
#
# print table and class-specific accuracy
cbind(tab, cl_acc = diag(tab)/rowSums(tab))
```

```
##                    asc_stairs basketball cross_trainer cycling_horiz
## asc_stairs                 80          0             0             0
## basketball                  0         75             0             0
## cross_trainer               0          0            78             0
## cycling_horiz               0          0             0            78
## cycling_vert                0          0             0             0
## desc_stairs                 3          0             0             0
## jumping                     0          0             0             0
## lying_back                  0          0             0             0
## lying_side                  0          0             0             0
## moving_elevator             1          0             0             0
## rowing                      0          0             0             0
## running_treadmill           0          0             0             0
## sitting                     0          0             0             0
## stand_elevator              0          0             0             0
## standing                    0          0             0             0
## stepper                     0          0             0             0
## walking                    10          0             0             0
## walking_tread               0          0             0             0
## walking_tread_incl          0          0             0             0
##                    cycling_vert desc_stairs jumping lying_back lying_side
## asc_stairs                    0           0       0          0          0
## basketball                    0           0       1          0          0
## cross_trainer                 0           0       0          0          0
## cycling_horiz                 0           0       0          0          0
## cycling_vert                 77           0       0          0          0
## desc_stairs                   0          76       0          0          0
## jumping                       0           0      80          0          0
## lying_back                    0           0       0         79          0
## lying_side                    0           0       0          0         80
## moving_elevator               0           0       0          0          0
## rowing                        0           0       0          0          0
## running_treadmill             0           0       0          0          0
## sitting                       0           0       0          0          0
## stand_elevator                0           0       0          0          0
## standing                      0           0       0          0          0
## stepper                       0           0       0          0          0
```

```
## walking                        0           2       0          0          0
## walking_tread                  0           0       0          0          0
## walking_tread_incl             0           0       0          0          0
##                    moving_elevator rowing running_treadmill sitting
## asc_stairs                       0      0                  0       0
## basketball                       3      0                  0       0
## cross_trainer                    0      0                  0       0
## cycling_horiz                    0      1                  0       1
## cycling_vert                     0      0                  0       0
## desc_stairs                      0      0                  0       0
## jumping                          0      0                  0       0
## lying_back                       0      1                  0       0
## lying_side                       0      0                  0       0
## moving_elevator                 55      0                  0       0
## rowing                           0     80                  0       0
## running_treadmill                0      0                 80       0
## sitting                          0      0                  0      80
## stand_elevator                   1      0                  0       0
## standing                         1      0                  0       0
## stepper                          0      0                  0       0
## walking                          1      0                  0       0
## walking_tread                    0      0                  0       0
## walking_tread_incl               0      0                  0       0
##                    stand_elevator standing stepper walking walking_tread
## asc_stairs                      0        0       0       0             0
## basketball                      0        0       1       0             0
## cross_trainer                   0        0       2       0             0
## cycling_horiz                   0        0       0       0             0
## cycling_vert                    0        0       3       0             0
## desc_stairs                     0        0       0       1             0
## jumping                         0        0       0       0             0
## lying_back                      0        0       0       0             0
## lying_side                      0        0       0       0             0
## moving_elevator                 9        9       4       1             0
## rowing                          0        0       0       0             0
## running_treadmill               0        0       0       0             0
## sitting                         0        0       0       0             0
## stand_elevator                 65       14       0       0             0
## standing                        1       78       0       0             0
## stepper                         0        0      80       0             0
## walking                         0        0       0      57             0
## walking_tread                   0        0       0       0            75
## walking_tread_incl              0        0       0       0             0
##                    walking_tread_incl cl_acc
## asc_stairs                          0 1.0000
## basketball                          0 0.9375
## cross_trainer                       0 0.9750
## cycling_horiz                       0 0.9750
## cycling_vert                        0 0.9625
## desc_stairs                         0 0.9500
```

```
## jumping                              0 1.0000
## lying_back                           0 0.9875
## lying_side                           0 1.0000
## moving_elevator                      1 0.6875
## rowing                               0 1.0000
## running_treadmill                    0 1.0000
## sitting                              0 1.0000
## stand_elevator                       0 0.8125
## standing                             0 0.9750
## stepper                              0 1.0000
## walking                             10 0.7125
## walking_tread                        5 0.9375
## walking_tread_incl                  80 1.0000
```

```
# find the 1st, 2nd and 3rd most likely class
ranks <- t( apply(tab, 1, function(x) {
class_labels[ order(x, decreasing = TRUE)[1:4]] } ) )
ranks
```

```
##
## class_y              [,1]                [,2]
##   asc_stairs         "asc_stairs"        "basketball"
##   basketball         "basketball"        "moving_elevator"
##   cross_trainer      "cross_trainer"     "stepper"
##   cycling_horiz      "cycling_horiz"     "rowing"
##   cycling_vert       "cycling_vert"      "stepper"
##   desc_stairs        "desc_stairs"       "asc_stairs"
##   jumping            "jumping"           "asc_stairs"
##   lying_back         "lying_back"        "rowing"
##   lying_side         "lying_side"        "asc_stairs"
##   moving_elevator    "moving_elevator"   "stand_elevator"
##   rowing             "rowing"            "asc_stairs"
##   running_treadmill  "running_treadmill" "asc_stairs"
##   sitting            "sitting"           "asc_stairs"
##   stand_elevator     "stand_elevator"    "standing"
##   standing           "standing"          "moving_elevator"
##   stepper            "stepper"           "asc_stairs"
##   walking            "walking"           "asc_stairs"
##   walking_tread      "walking_tread"     "walking_tread_incl"
##   walking_tread_incl "walking_tread_incl" "asc_stairs"
##
## class_y              [,3]                [,4]
##   asc_stairs         "cross_trainer"     "cycling_horiz"
##   basketball         "jumping"           "stepper"
##   cross_trainer      "asc_stairs"        "basketball"
##   cycling_horiz      "sitting"           "asc_stairs"
##   cycling_vert       "asc_stairs"        "basketball"
##   desc_stairs        "walking"           "basketball"
##   jumping            "basketball"        "cross_trainer"
##   lying_back         "asc_stairs"        "basketball"
```

```
##   lying_side         "basketball"         "cross_trainer"
##   moving_elevator    "standing"           "stepper"
##   rowing             "basketball"         "cross_trainer"
##   running_treadmill  "basketball"         "cross_trainer"
##   sitting            "basketball"         "cross_trainer"
##   stand_elevator     "moving_elevator"    "asc_stairs"
##   standing           "stand_elevator"     "asc_stairs"
##   stepper            "basketball"         "cross_trainer"
##   walking            "walking_tread_incl" "desc_stairs"
##   walking_tread      "asc_stairs"         "basketball"
##   walking_tread_incl "basketball"         "cross_trainer"
```