Configuration and Intercomparison of Deep Learning Neural Models for Statistical Downscaling (Precipitation)

J. Baño-Medina, R. Manzanas and J.M Gutiérrez

2019-07-03

Contents

1 Loading Data

We load the core libraries of climate4R: loadeR, transformeR, downscaleR and visualizeR. We also load climate4R.value of climate4R which would permit us to compute the validation indices as well as other auxiliary libraries mainly for plotting concerns.

```
library(loadeR)
library(transformeR)
library(downscaleR)
library(visualizeR)
library(climate4R.value)
library(magrittr)
library(gridExtra)
library(RColorBrewer)
library(sp)
```

To access the datasets we first log in to our UDG account:

```
loginUDG(username = "", password = "")
```

We find the label associated to ERA-Interim via the UDG.datasets() function of loadeR: "ECMWF_ERA-Interim-ESD". Then we load the predictors by calling loadGridData of loadeR.

```
variables <- c("z@500","z@700","z@850","z@1000",
               "hus@500","hus@700","hus@850","hus@1000",
               "ta@500","ta@700","ta@850","ta@1000",
               "ua@500","ua@700","ua@850","ua@1000",
               "va@500","va@700","va@850","va@1000")
x <- lapply(variables, function(x) {
  loadGridData(dataset = "ECMWF_ERA-Interim-ESD",
               var = x,
```

```
              lonLim = c(-10,32), # 22 puntos en total
              latLim = c(36,72),  # 19 puntos en total
              years = 1979:2008)
}) %>% makeMultiGrid()
```

The E-OBS dataset is also accesible in the UDG datasets. Thus, we load the predictand dataset by calling again loadGridData.

```
y <- loadGridData(dataset = "E-OBS_v14_0.50regular",
                  var = "pr",lonLim = c(-10,32),
                  latLim = c(36,72),
                  years = 1979:2008)
```

We split into train (i.e., 1979-2002) and test (i.e., 2003-2008).

```
# Train
xT <- subsetGrid(x,years = 1979:2002)
yT <- subsetGrid(y,years = 1979:2002)
yT_bin <- binaryGrid(yT,threshold = 1,condition = "GT")
# Test
xt <- subsetGrid(x,years = 2003:2008)
yt <- subsetGrid(y,years = 2003:2008)
yt_bin <- binaryGrid(yt,threshold = 1,condition = "GT")

save(xT,file = "./Data/precip/xT.rda")
save(xt,file = "./Data/precip/xt.rda")
save(yT,file = "./Data/precip/yT.rda")
save(yt,file = "./Data/precip/yt.rda")
save(yT_bin,file = "./Data/precip/yT_bin.rda")
save(yt_bin,file = "./Data/precip/yt_bin.rda")
rm(x,y)
```

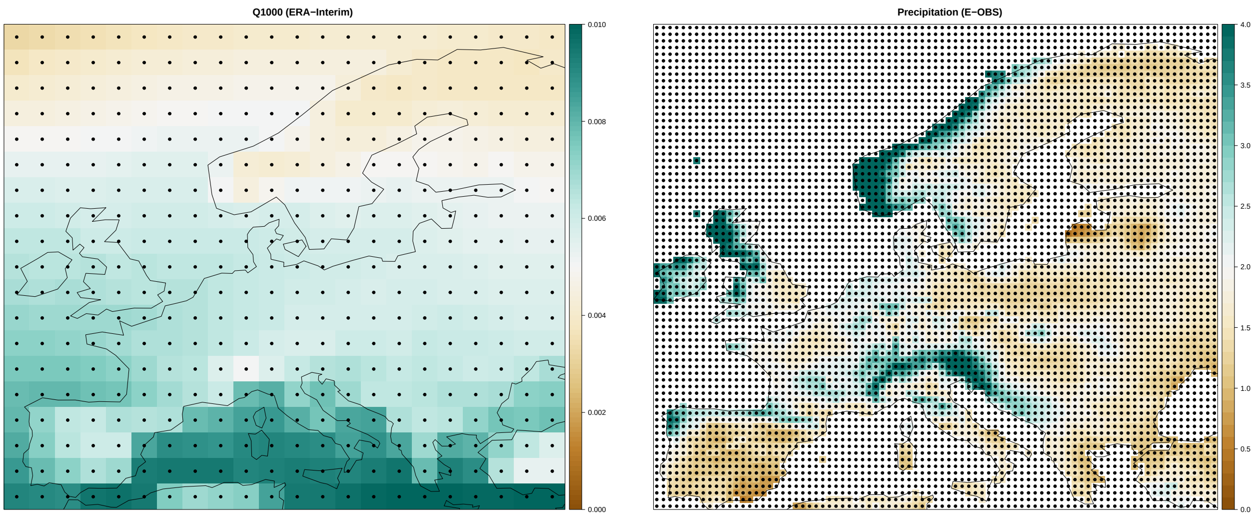
We can take a look at the grid resolutions of ERA-Interim and E-OBS in order to better visualize the gap we try to bridge with the downscaling.

```
colsindex <- brewer.pal(n = 9, "BrBG")
cb <- colorRampPalette(colsindex)
coords_x <- expand.grid(xt$xyCoords$x,xt$xyCoords$y) ; names(coords_x) <- c("x","y")
coords_y <- expand.grid(yt$xyCoords$x,yt$xyCoords$y) ; names(coords_y) <- c("x","y")

pplot <- list()
pplot[[1]] <- spatialPlot(climatology(subsetGrid(xt,var = "hus@1000")), backdrop.theme = "coastline",
                          main = "Q1000 (ERA-Interim)",
                          col.regions = cb,
                          at = seq(0,0.01, 0.0001),
                          set.min = 0, set.max = 0.01, colorkey = TRUE,
                          sp.layout = list(list(SpatialPoints(coords_x),
                                                first = FALSE, col = "black",
                                                pch = 20, cex = 1)))
pplot[[2]] <- spatialPlot(climatology(yt), backdrop.theme = "coastline",
                          main = "Precipitation (E-OBS)",
                          col.regions = cb,
                          at = seq(0, 4, 0.1),
                          set.min = 0, set.max = 4, colorkey = TRUE,
                          sp.layout = list(list(SpatialPoints(coords_y),
                                                first = FALSE, col = "black",
```

```
                                                  pch = 20, cex = 1)))

lay = rbind(c(1,2))
grid.arrange(grobs = pplot, layout_matrix = lay)
```



We can visualize some statistics of the train and test distributions, such as the climatology, the frequency of rainy days and the percentile 98th in order to gain knowledge about the observed data. To compute the statistics we use the library climate4R.value of climate4R.

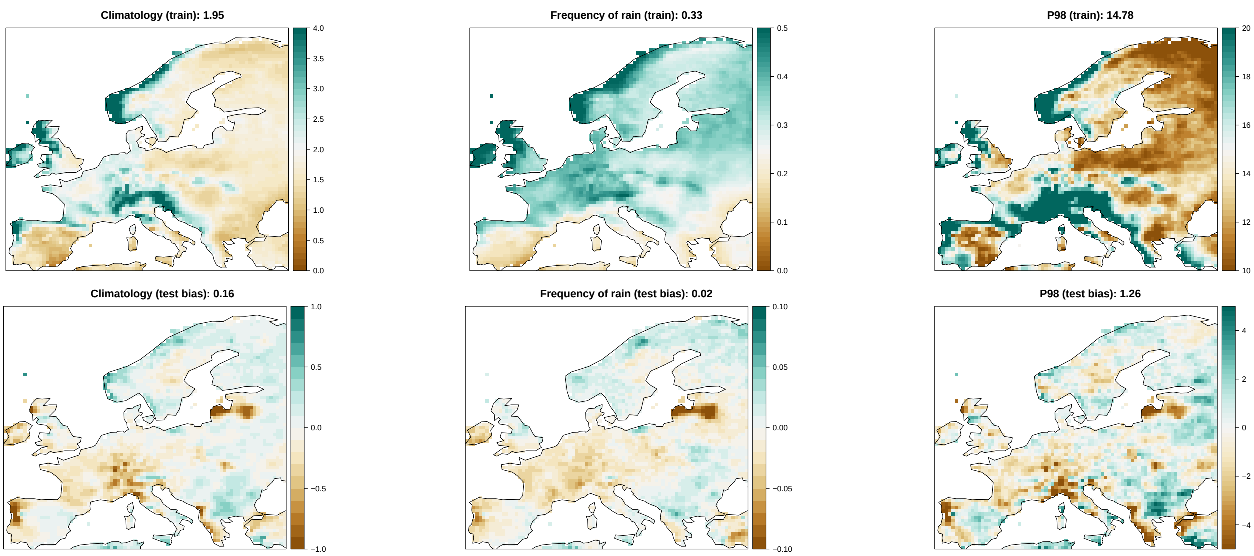
```
colsindex <- brewer.pal(n = 9, "BrBG")
cb <- colorRampPalette(colsindex)

pplot <- at <- list()
n1 <- 0; n2 <- 3
indexNames <- c("Climatology", "Frequency of rain", "P98")
for (indexName in indexNames) {
  if (indexName == "Climatology") {
    indexTrain <- valueIndex(yT,index.code = "Mean")$Index %>% redim()
    indexTest <- valueIndex(yt,index.code = "Mean")$Index %>% redim()
    at[[1]] <- seq(0, 4, 0.1); at[[2]] <- seq(-1, 1, 0.1)
  }
  if (indexName == "Frequency of rain") {
    indexTrain <- valueIndex(yT_bin,index.code = "Mean")$Index %>% redim()
    indexTest <- valueIndex(yt_bin,index.code = "Mean")$Index %>% redim()
    at[[1]] <- seq(0, 0.5, 0.01); at[[2]] <- seq(-0.1, 0.1, 0.01)
  }
  if (indexName == "P98") {
    indexTrain <- valueIndex(yT,index.code = "P98")$Index %>% redim()
    indexTest <- valueIndex(yt,index.code = "P98")$Index %>% redim()
    at[[1]] <- seq(10, 20, 0.25); at[[2]] <- seq(-5, 5, 0.2)
  }

  for (i in 1:2) {
    if (i == 1) {
      dataset <- "(train)"; index <- indexTrain; n1 <- n1 + 1; n <- n1
      }
    if (i == 2) {
      indexTest <- gridArithmetics(indexTest,indexTrain,operator = "-")
```

```
      dataset <- "(test bias)"; index <- indexTest; n2 <- n2 + 1; n <- n2
      }
    pplot[[n]] <- spatialPlot(climatology(index), backdrop.theme = "coastline",
                              main = paste(indexName,paste0(dataset,":"),
                                           round(mean(abs(index$Data), na.rm = TRUE),digits = 2)),
                              col.regions = cb,
                              at = at[[i]],
                              set.min = at[[i]][1], set.max = at[[i]][length(at[[i]])],
                              colorkey = TRUE)
  }
}

lay = rbind(c(1,2,3),
            c(4,5,6))
grid.arrange(grobs = pplot, layout_matrix = lay)
```



Once the data is loaded we standardize the predictors by calling scaleGrid function of transformeR.

```
xt <- scaleGrid(xt,xT, type = "standardize", spatial.frame = "gridbox") %>% redim(drop = TRUE)
xT <- scaleGrid(xT, type = "standardize", spatial.frame = "gridbox") %>% redim(drop = TRUE)
```

2 Downscaling

2.1 Generalized Linear Models (GLM)

We define a function that encapsulates downscaleChunk, which is the function of downscaleR that calls the glm function. Therefore, this function downscales to the predictand resolution and saves the prediction. It has to be noticed that the downscaling of precipitation ocurs at two stages: the ocurrence of precipitation and the amount of precipitation.

```
trainPredictGLM <- function(x,y,newdata,neighbours=1,filename) {
  y.ocu <- binaryGrid(y,condition = "GT",threshold = 1)
  y.rest <- gridArithmetics(y,1,operator = "-")

  # Logistic Regression (DETERMINISTIC)
```

```r
downscaleChunk(x = x, y = y.ocu, newdata = newdata,
               method = "GLM", family = binomial(link = "logit"),
               prepareData.args = list(local.predictors = list(n=neighbours, vars = getVarNames(x)))
)
lf <- list.files("./", pattern = "dataset1", full.names = TRUE)
chunk.list <- lapply(lf, function(x) get(load(x)))
predDET_ocu_train <- bindGrid(chunk.list, dimension = "lat")
file.remove(lf)
lf <- list.files("./", pattern = "dataset2", full.names = TRUE)
chunk.list <- lapply(lf, function(x) get(load(x)))
predDET_ocu_test <- bindGrid(chunk.list, dimension = "lat")
file.remove(lf)

# Logistic Regression (STOCHASTIC)
downscaleChunk(x = x, y = y.ocu, newdata = newdata,
               method = "GLM", family = binomial(link = "logit"),simulate = TRUE,
               prepareData.args = list(local.predictors = list(n=neighbours, vars = getVarNames(x)))
)
lf <- list.files("./", pattern = "dataset1", full.names = TRUE)
chunk.list <- lapply(lf, function(x) get(load(x)))
predSTO_ocu_train <- bindGrid(chunk.list, dimension = "lat")
file.remove(lf)
lf <- list.files("./", pattern = "dataset2", full.names = TRUE)
chunk.list <- lapply(lf, function(x) get(load(x)))
predSTO_ocu_test <- bindGrid(chunk.list, dimension = "lat")
file.remove(lf)

# Gamma Regression with link logarithmic (DETERMINISTIC)
downscaleChunk(x = x, y = y.rest, newdata = newdata,
               method = "GLM", family = Gamma(link = "log"), condition = "GT", threshold = 0,
               prepareData.args = list(local.predictors = list(n=neighbours, vars = getVarNames(x)))
)
lf <- list.files("./", pattern = "dataset1", full.names = TRUE)
chunk.list <- lapply(lf, function(x) get(load(x)))
predDET_reg_train <- bindGrid(chunk.list, dimension = "lat") %>% gridArithmetics(1,operator = "+")
file.remove(lf)
lf <- list.files("./", pattern = "dataset2", full.names = TRUE)
chunk.list <- lapply(lf, function(x) get(load(x)))
predDET_reg_test <- bindGrid(chunk.list, dimension = "lat") %>% gridArithmetics(1,operator = "+")
file.remove(lf)

# Gamma Regression with link logarithmic (STOCHASTIC)
downscaleChunk(x = x, y = y.rest, newdata = newdata,
               method = "GLM", family = Gamma(link = "log"), simulate = TRUE,
               condition = "GT", threshold = 0,
               prepareData.args = list(local.predictors = list(n=neighbours, vars = getVarNames(x)))
)
lf <- list.files("./", pattern = "dataset1", full.names = TRUE)
chunk.list <- lapply(lf, function(x) get(load(x)))
predSTO_reg_train <- bindGrid(chunk.list, dimension = "lat") %>% gridArithmetics(1,operator = "+")
file.remove(lf)
lf <- list.files("./", pattern = "dataset2", full.names = TRUE)
chunk.list <- lapply(lf, function(x) get(load(x)))
```

```
  predSTO_reg_test <- bindGrid(chunk.list, dimension = "lat") %>% gridArithmetics(1,operator = "+")
  file.remove(lf)

  ###
  predDET_ocu <- predDET_ocu_train %>% redim(drop = TRUE)
  predDET_reg <- predDET_reg_train %>% redim(drop = TRUE)
  predSTO_ocu <- predSTO_ocu_train %>% redim(drop = TRUE)
  predSTO_reg <- predSTO_reg_train %>% redim(drop = TRUE)
  save(predDET_ocu,predDET_reg,
       predSTO_ocu,predSTO_reg,
       file = paste0("./Data/precip/predictions_train_",filename,".rda"))

  predDET_ocu <- predDET_ocu_test %>% redim(drop = TRUE)
  predDET_reg <- predDET_reg_test %>% redim(drop = TRUE)
  predSTO_ocu <- predSTO_ocu_test %>% redim(drop = TRUE)
  predSTO_reg <- predSTO_reg_test %>% redim(drop = TRUE)
  save(predDET_ocu,predDET_reg,
       predSTO_ocu,predSTO_reg,
       file = paste0("./Data/precip/predictions_test_",filename,".rda"))
}
```

Now, we downscale by calling the function defined above.

```
# glm1
trainPredictGLM(x = xT,y = yT,newdata = list(xt),
                      filename = "glm1",neighbours = 1)
# glm4
trainPredictGLM(x = xT,y = yT,newdata = list(xt),
                      filename = "glm4",neighbours = 4)
```

2.2 Downscaling - Deep Neural Networks

2.2.1 Training

To infer deep learning models we rely on Keras.

```
library(keras)
```

In this work, all deep learning functions have optimized the negative log-likelihood of a Bernouilli-Gamma distribution. We implement the custom loss in Keras.

```
discreteGamma <- custom_metric("custom_loss", function(true, pred){
  K <- backend()
  D <- K$int_shape(pred)[[2]]/3
  ocurrence = pred[,1:D]
  shape_parameter = K$exp(pred[,(D+1):(D*2)])
  scale_parameter = K$exp(pred[,(D*2+1):(D*3)])
  bool_rain = K$cast(K$greater(true,0),K$tf$float32)
  epsilon = 0.000001
  return (- K$mean((1-bool_rain)*K$tf$log(1-ocurrence+epsilon) + bool_rain*(K$tf$log(ocurrence+epsilon)
})
```

In the following we provide the code related to the deep learning arquitechtures.

```r
arquitechtures <- function(arquitechture,input_shape,output_shape) {
  if (arquitechture == "CNN1lin") {
    inputs <- layer_input(shape = input_shape)
    x = inputs
    l1 = layer_conv_2d(x ,filters = 50, kernel_size = c(3,3), activation = 'linear', padding = "same")
    l2 = layer_conv_2d(l1,filters = 25, kernel_size = c(3,3), activation = 'linear', padding = "same")
    l3 = layer_conv_2d(l2,filters = 1, kernel_size = c(3,3), activation = 'linear', padding = "same")
    l4 = layer_flatten(l3)
    parameter1 = layer_dense(l4,units = output_shape, activation = "sigmoid")
    parameter2 = layer_dense(l4,units = output_shape)
    parameter3 = layer_dense(l4,units = output_shape)
    outputs = layer_concatenate(list(parameter1,parameter2,parameter3))
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  if (arquitechture == "CNN1") {
    inputs <- layer_input(shape = input_shape)
    x = inputs
    l1 = layer_conv_2d(x ,filters = 50, kernel_size = c(3,3), activation = 'relu', padding = "same")
    l2 = layer_conv_2d(l1,filters = 25, kernel_size = c(3,3), activation = 'relu', padding = "same")
    l3 = layer_conv_2d(l2,filters = 1, kernel_size = c(3,3), activation = 'relu', padding = "same")
    l4 = layer_flatten(l3)
    parameter1 = layer_dense(l4,units = output_shape, activation = "sigmoid")
    parameter2 = layer_dense(l4,units = output_shape)
    parameter3 = layer_dense(l4,units = output_shape)
    outputs = layer_concatenate(list(parameter1,parameter2,parameter3))
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  if (arquitechture == "CNN10") {
    inputs <- layer_input(shape = input_shape)
    x = inputs
    l1 = layer_conv_2d(x ,filters = 50, kernel_size = c(3,3), activation = 'relu', padding = "valid")
    l2 = layer_conv_2d(l1,filters = 25, kernel_size = c(3,3), activation = 'relu', padding = "valid")
    l3 = layer_conv_2d(l2,filters = 10, kernel_size = c(3,3), activation = 'relu', padding = "valid")
    l4 = layer_flatten(l3)
    parameter1 = layer_dense(l4,units = output_shape, activation = "sigmoid")
    parameter2 = layer_dense(l4,units = output_shape)
    parameter3 = layer_dense(l4,units = output_shape)
    outputs = layer_concatenate(list(parameter1,parameter2,parameter3))
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  if (arquitechture == "CNNinverse") {
    inputs <- layer_input(shape = input_shape)
    x = inputs
    l1 = layer_conv_2d(x ,filters = 10, kernel_size = c(3,3), activation = 'relu', padding = "valid")
    l2 = layer_conv_2d(l1,filters = 25, kernel_size = c(3,3), activation = 'relu', padding = "valid")
    l3 = layer_conv_2d(l2,filters = 50, kernel_size = c(3,3), activation = 'relu', padding = "valid")
    l4 = layer_flatten(l3)
    parameter1 = layer_dense(l4,units = output_shape, activation = "sigmoid")
    parameter2 = layer_dense(l4,units = output_shape)
    parameter3 = layer_dense(l4,units = output_shape)
```

```
    outputs = layer_concatenate(list(parameter1,parameter2,parameter3))
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  if (arquitechture == "CNNdense") {
    inputs <- layer_input(shape = input_shape)
    x = inputs
    l1 = layer_conv_2d(x ,filters = 50, kernel_size = c(3,3), activation = 'relu', padding = "valid")
    l2 = layer_conv_2d(l1,filters = 25, kernel_size = c(3,3), activation = 'relu', padding = "valid")
    l3 = layer_conv_2d(l2,filters = 10, kernel_size = c(3,3), activation = 'relu', padding = "valid")
    l4 = layer_flatten(l3)
    l5 = layer_dense(l4,units = 50, activation = "relu")
    l6 = layer_dense(l5,units = 50, activation = "relu")
    parameter1 = layer_dense(l6,units = output_shape, activation = "sigmoid")
    parameter2 = layer_dense(l6,units = output_shape)
    parameter3 = layer_dense(l6,units = output_shape)
    outputs = layer_concatenate(list(parameter1,parameter2,parameter3))
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  return(model)
}
```

To train the latter arquitechtures we have encapsulated them into a more general funcion called trainDEEP.

```
trainDEEP <- function(x,y,arquitechture, epochs=10000,patience=30,
                      learning_rate = 0.0001,loss_function = discreteGamma){

  x <- x$Data
  x <- x %>% aperm(c(2,3,4,1))
  y <- y$Data
  dim(y) <- c(dim(y)[1],dim(y)[2]*dim(y)[3])
  indLand <- (!apply(y,MARGIN = 2,anyNA)) %>% which()
  y <- y[,indLand]
  y <- y - 1
  y[which(y < 0, arr.ind = TRUE)] <- 0

  callbacks <- list(callback_early_stopping(patience = patience),
                    callback_model_checkpoint(filepath=paste0('./models/precip/',arquitechture,'.h5'),
                                              monitor='val_loss', save_best_only=TRUE)
  )

  model <- arquitechtures(arquitechture,input_shape = dim(x)[-1],output_shape = ncol(y))
  model %>% compile(optimizer = optimizer_adam(lr = learning_rate), loss = loss_function)
  model %>% fit(x, y, epochs = epochs, batch_size = 100,
                validation_split = 0.1, callbacks = callbacks, verbose = 0)
  k_clear_session()
}
```

Finally, we train the models and save the model in the path specified in the trainDEEP function, according to the early-stopping criteria.

```
trainDEEP(xT,yT,arquitechture = "CNN1lin")
trainDEEP(xT,yT,arquitechture = "CNN1")
trainDEEP(xT,yT,arquitechture = "CNN10")
```

```r
trainDEEP(xT,yT,arquitechture = "CNNinverse")
trainDEEP(xT,yT,arquitechture = "CNNdense")
```

2.2.2 Prediction

Once the models are trained and saved, we predict onto the train and test datasets. To do so, we first define a function called predictDEEP that encapsulates the deterministic and stochastic predictions. Recall that, accpording to the loss function, the net estimates the parameters p, α and β of a Bernouilli-Gamma distribution. In the deterministic way, the prediction for a given day is the expectance of the conditional Bernouilli-Gamma distribution infered. In the stochastic way, we sample from the conditional Bernouilli-Gamma distribution. This sampling is coded in the functions simulateOcu and simulateReg.

```r
predictDEEP <- function(x,template,arquitechture,dataset) {
  loss_function <- discreteGamma
  model <- load_model_hdf5(filepath = paste0("./models/precip/",arquitechture,".h5"),
                           custom_objects = c("custom_loss" = loss_function))
  x <- x$Data %>% aperm(c(2,3,4,1))
  pred <- model$predict(x)
  D <- ncol(pred)/3
  ntime <- dim(template$Data)[1]
  nlat <- dim(template$Data)[2]
  nlon <- dim(template$Data)[3]

  # Deterministic Prediction
  predDET_ocu <- pred[,1:D]
  predDET_reg <- exp(pred[,(D+1):(D*2)])*exp(pred[,(D*2+1):(D*3)]) + 1

  # Stochastic Prediction
  simulate_ocu <- function(dat,model,D){
    ocu <- model$predict(dat)[,1:D,drop = FALSE]
    sim <- matrix(runif(length(ocu),min = 0,max = 1), nrow = nrow(ocu), ncol = ncol(ocu))
    cond <- ocu > sim
    return(cond*1)
  }

  simulate_reg <- function(dat,model,D) {
    shape <- exp(model$predict(dat)[,(D+1):(D*2),drop = FALSE])
    scale <- exp(model$predict(dat)[,(D*2+1):(D*3),drop = FALSE])
    p <- matrix(nrow = nrow(dat),ncol = D)
    for (i in 1:D) {
      p[,i] <- rgamma(n = nrow(dat), shape = shape[,i], scale = scale[,i])
    }
    return(p)
  }
  predSTO_ocu <- simulate_ocu(x,model,D)
  predSTO_reg <- simulate_reg(x,model,D) + 1

  # Converting to 2D-map (including sea)
  yT1D <- yT$Data
  dim(yT1D) <- c(dim(yT1D)[1],nlat*nlon)
  indLand <- (!apply(yT1D,MARGIN = 2,anyNA)) %>% which()
  convert2map2D <- function(grid,template) {
    dim(template$Data) <- c(ntime,nlat*nlon)
```

```r
    out <- template
    out$Data[,indLand] <- grid
    dim(out$Data) <- c(ntime,nlat,nlon)
    return(out)
  }

  predDET_ocu <- convert2map2D(predDET_ocu,template)
  predDET_reg <- convert2map2D(predDET_reg,template)
  predSTO_ocu <- convert2map2D(predSTO_ocu,template)
  predSTO_reg <- convert2map2D(predSTO_reg,template)

  save(predDET_ocu,predDET_reg,predSTO_ocu,predSTO_reg,
       file = paste0("./Data/precip/predictions_",dataset,"_",arquitechture,".rda"))
  k_clear_session()
}
```

Then we predict and save the predictions.

```r
# Train
predictDEEP(xT,arquitechture = "CNN1lin",template = yT,dataset = "train")
predictDEEP(xT,arquitechture = "CNN1",template = yT,dataset = "train")
predictDEEP(xT,arquitechture = "CNN10",template = yT,dataset = "train")
predictDEEP(xT,arquitechture = "CNNinverse",template = yT,dataset = "train")
predictDEEP(xT,arquitechture = "CNNdense",template = yT,dataset = "train")

# Test
predictDEEP(xt,arquitechture = "CNN1lin",template = yt,dataset = "test")
predictDEEP(xt,arquitechture = "CNN1",template = yt,dataset = "test")
predictDEEP(xt,arquitechture = "CNN10",template = yt,dataset = "test")
predictDEEP(xt,arquitechture = "CNNinverse",template = yt,dataset = "test")
predictDEEP(xt,arquitechture = "CNNdense",template = yt,dataset = "test")
```

3 Validation of the Results

3.1 Intercomparison of Methods

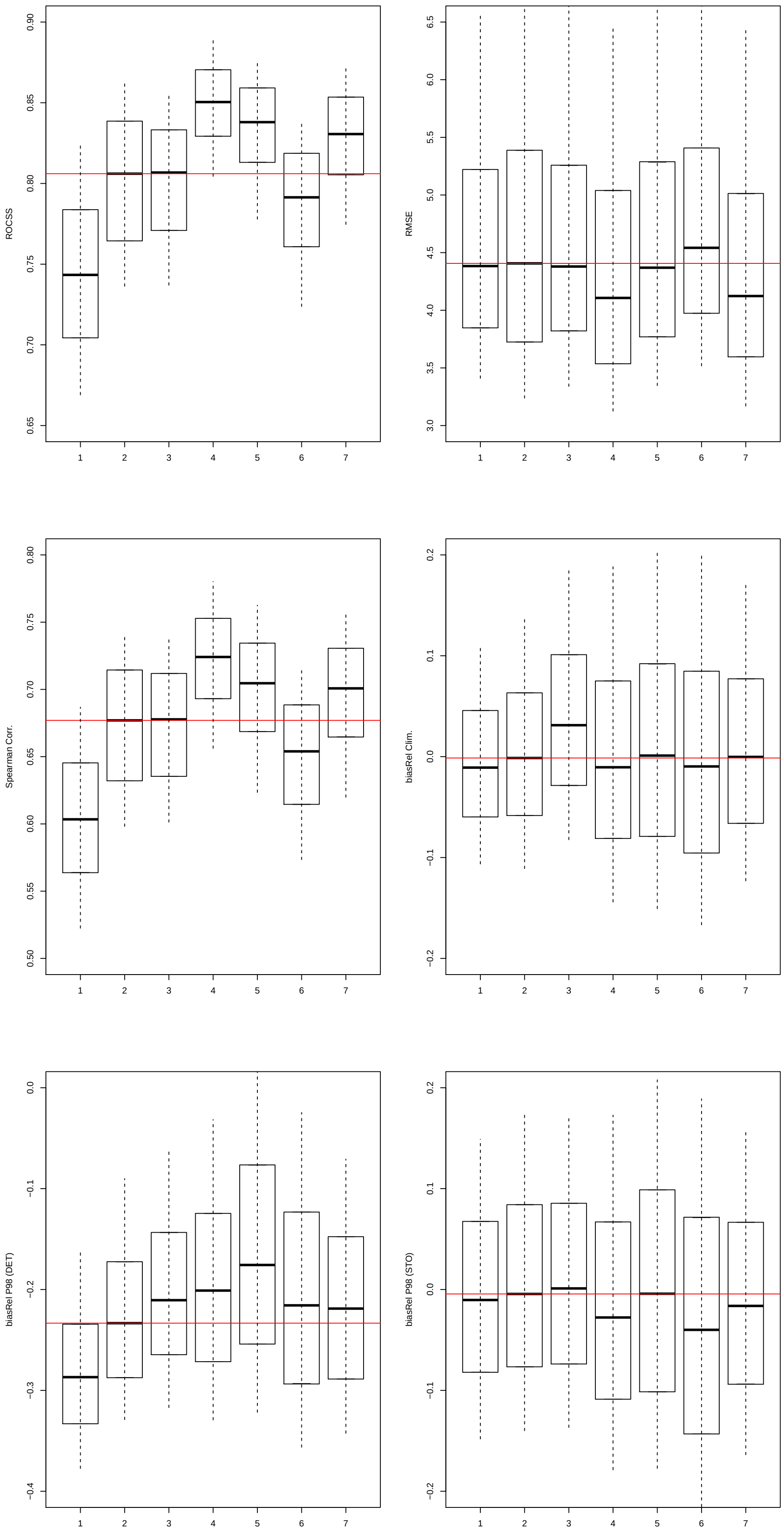
In this figure, we calculate the validation indices by using the library climate4R.value of climate4R. In particular the indices used are: the Roc Skill Score (ROCSS), the Root Mean Squared Error (RMSE), the spearman correlation and the relative biases of the climatology and of the percentile 98 of the rainy days distribution (both deterministic and stochastic predictions). The indices are plotted with the function violinPlot of climate4R's package visualizeR.

```r
# file_name <- paste0("./figures/fig04_precipitation.pdf")
# pdf(file = file_name,width = 5,height = 10)
par(mfrow = c(3,2))

filepreds=list("glm1","glm4",
               "CNN1lin","CNN1","CNN10",
               "CNNinverse","CNNdense")
datasetspred <- length(filepreds)
indexNames <- c("ROCSS","RMSE","Spearman Corr.",
                "biasRel Clim.","biasRel P98 (DET)","biasRel P98 (STO)")
pplot <- list(); n <- 0
```

```r
for (indexName in indexNames) {
  index <- array(dim = c(datasetspred,dim(yT$Data)[2:3]))
  for (i in 1:datasetspred) {
    load(paste0("./Data/precip/predictions_train_",filepreds[[i]],".rda"))
    predDET_ocu_train <- predDET_ocu
    load(paste0("./Data/precip/predictions_test_",filepreds[[i]],".rda"))
    predDET_bin <- binaryGrid(predDET_ocu,ref.obs = yT_bin,ref.pred = predDET_ocu_train)
    predDET <- gridArithmetics(predDET_bin,predDET_reg,operator = "*")
    predSTO <- gridArithmetics(predSTO_ocu,predSTO_reg,operator = "*")

    if (indexName == "ROCSS") {
      index[i,,] <- valueMeasure(yt_bin,predDET_ocu,
                                 measure.code="ts.rocss")$Measure$Data
      ylim <- c(0.65,0.9)
    }
    if (indexName == "RMSE") {
      index[i,,] <- valueMeasure(yt,predDET_reg,measure.code="ts.RMSE",
                                 condition = "GT", threshold = 1,
                                 which.wetdays = "Observation")$Measure$Data
      ylim <- c(3,6.5)
    }
    if (indexName == "Spearman Corr.") {
      index[i,,] <- valueMeasure(yt,predDET,measure.code="ts.rs")$Measure$Data
      ylim <- c(0.5,0.8)
    }
    if (indexName == "biasRel Clim.") {
      index[i,,] <- valueMeasure(yt,predDET,measure.code="biasRel",index.code="Mean")$Measure$Data
      ylim <- c(-0.2,0.2)
    }
    if (indexName == "biasRel P98 (DET)") {
      index[i,,] <- valueMeasure(yt,predDET,measure.code="biasRel",index.code="P98")$Measure$Data
      ylim <- c(-0.4,0.0)
    }
    if (indexName == "biasRel P98 (STO)") {
      index[i,,] <- valueMeasure(yt,predSTO,measure.code="biasRel",index.code="P98")$Measure$Data
      ylim <- c(-0.2,0.2)
    }
  }
  n <- n + 1
  dim(index) <- c(datasetspred,prod(dim(yT$Data)[2:3]))
  indLand <- (!apply(index,MARGIN = 2,anyNA)) %>% which()
  index <- index[,indLand] %>% t()
  mglm4 <- median(index[,2],na.rm = TRUE)
  perc <- apply(index,MARGIN = 2,FUN = function(z) quantile(z,probs = c(0.1,0.9)))
  boxplot(index, outline = FALSE, asp = 1, ylim = ylim, range = 0.0001, ylab = indexName, asp = 1)
  lines(c(0,8),c(mglm4,mglm4), col = "red", asp = 1)
  for (i in 1:datasetspred) lines(c(i,i),perc[,i], lty = 2)
}
```


ROCSS
RMSE
Spearman Corr.
biasRel Clim.
biasRel P98 (DET)
biasRel P98 (STO)

```r
# dev.off()
```

3.2 Spatial Maps

The above indices can be visualized spatially, this is, observing the results per gridbox. To do so, we define a function called experiment1 that encapsulates the code needed for plotting. The spatial map is plotted with the function spatialPlot of visualizeR, which we remind is a package of climate4R.

```r
experiment1 <- function(model) {
  cb <- colorRampPalette(brewer.pal(9, "OrRd"))(80)
  colsindex <- rev(brewer.pal(n = 9, "RdBu"))
  cb2 <- colorRampPalette(colsindex)

  load(paste0("./Data/precip/predictions_train_",model,".rda"))
  predDET_ocu_train <- predDET_ocu
  load(paste0("./Data/precip/predictions_test_",model,".rda"))
  predDET_bin <- binaryGrid(predDET_ocu,ref.obs = yT_bin,ref.pred = predDET_ocu_train)
  predDET <- gridArithmetics(predDET_bin,predDET_reg,operator = "*")

  indexNames <- c("ROCSS","RMSE","Spearman Corr.",
                  "biasRel Clim.","biasRel P98 (DET)")

  pplot <- list(); n <- 0
  for (indexName in indexNames) {
    if (indexName == "ROCSS") {
      index <- valueMeasure(yt_bin,predDET_ocu,
                            measure.code="ts.rocss")$Measure %>% redim()
      at <- seq(0.5, 1, 0.01); colorbar <- cb
    }
    if (indexName == "RMSE") {
      index <- valueMeasure(yt,predDET_reg,
                            measure.code="ts.RMSE",
                            condition = "GT", threshold = 1,
                            which.wetdays = "Observation")$Measure %>% redim()
      at <- seq(2, 6.5, 0.25); colorbar <- cb
    }
    if (indexName == "Spearman Corr.") {
      index <- valueMeasure(yt,predDET,
                            measure.code="ts.rs")$Measure %>% redim()
      at <- seq(0.5, 1, 0.02); colorbar <- cb
    }
    if (indexName == "biasRel Clim.") {
      index <- valueMeasure(yt,predDET,
                            measure.code="biasRel",index.code="Mean")$Measure %>% redim()
      at <- seq(-0.5, 0.5, 0.01); colorbar <- cb2
    }
    if (indexName == "biasRel P98 (DET)") {
      index <- valueMeasure(yt,predDET,
                            measure.code="biasRel",index.code="P98")$Measure %>% redim()
       at <- seq(-0.5, 0.5, 0.01); colorbar <- cb2
    }

    n <- n + 1
```

```
    pplot[[n]] <- spatialPlot(climatology(index), backdrop.theme = "coastline",
                              main = paste(indexName,round(mean(abs(index$Data), na.rm = TRUE),
                                                           digits = 2)),
                              col.regions = colorbar,
                              at = at,
                              set.min = at[1], set.max = at[length(at)], colorkey = TRUE)
  }

  lay = rbind(c(1,2,3,4,5,6,7,8))
  grid.arrange(grobs = pplot, layout_matrix = lay)

}
```

Now we call the experiment1 to obtain the figures for the following methods: glm1, glm4 and CNN1.

```
figure <- list()
figure[[1]] <- experiment1(model = "glm1")
figure[[2]] <- experiment1(model = "glm4")
figure[[3]] <- experiment1(model = "CNN1")
```

```
lay = rbind(1,2,3)
grid.arrange(grobs = figure,layout_matrix = lay)
```