Configuration and Intercomparison of Deep Learning Neural Models for Statistical Downscaling

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

2019-12-19

Introduction

This notebook reproduces the results presented in the paper *Configuration and Intercomparison of Deep Learning Neural Models for Statistical Downscaling* by *J. Baño-Medina, R. Manzanas and J. M. Gutiérrez*, which has been submitted for discussion to *Geoscientific Model Development* in October 2019. In particular, the code developed herein concerns the downscaling of temperature and precipitation and therefore, their particularities are treated throughout the notebook in two different sections. Note that the programming lenguage is `R` and the technical specifications of the machine can be found at the end of the notebook.

Loading Data

All the working steps rely on the climate4R, which is a set of libraries especifically developed to handle climate data (loadeR, transformeR, downscaleR, visualizeR and climate4R.value). In this study, climate4R is used to load and post-process the data, downscaling, validation and visualization of the data. Especific notebooks have been developed to illustrate the working of climate4R functions and we refer the reader to the github repository that allocates these notebooks for more explanation on the particularities of every package and function.

Therefore, 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. To build deep learning models we rely on downscaleR.keras which integrates keras in the climate4R framework.

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

To load the data we rely on the loadeR package who permits an easy access to the datasets listed in the User Data Getaway (UDG) which is maintained by the Santander Meteorology Group. To access the datasets we first have to log in to our UDG account. If you do not have an account follow the instructions in the UDG description page

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

In order to avoid possible errors while running the notebook, you have to set the path to your desired working directory and create two files named "Data" and "models", that will contain the downscaled predictions and the trained deep models, respectively. Moreover, as we perform 2 distinct studies, one for precipitation and other for temperature, you should create 2 new directories named "precip" and "temperature" within the previous created directories (i.e., "Data" and "models"). An example of the latter would be "personalpath/Data/temperature". The predictions and models infered will be automatically saved in these folders and therefore not creating them will end into saving errors across the notebook.

```
path = ""
setwd(path)
dir.create("Data")
dir.create("Data/precip/")
dir.create("models")
dir.create("models/temperature/")
dir.create("models/precip/")
```

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()
```

Temperature

In this section we present the code needed to downscale temperature. Once the predictors were loaded above we proceed to download the predictand dataset: E-OBS version 14 at a resolution of 0.5º. The E-OBS dataset is also accesible in the UDG datasets. Thus, we load the temperature by calling again loadGridData.

```
y <- loadGridData(dataset = "E-OBS_v14_0.50regular",
                  var = "tas",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)
# Test
xt <- subsetGrid(x,years = 2003:2008)
yt <- subsetGrid(y,years = 2003:2008)
```

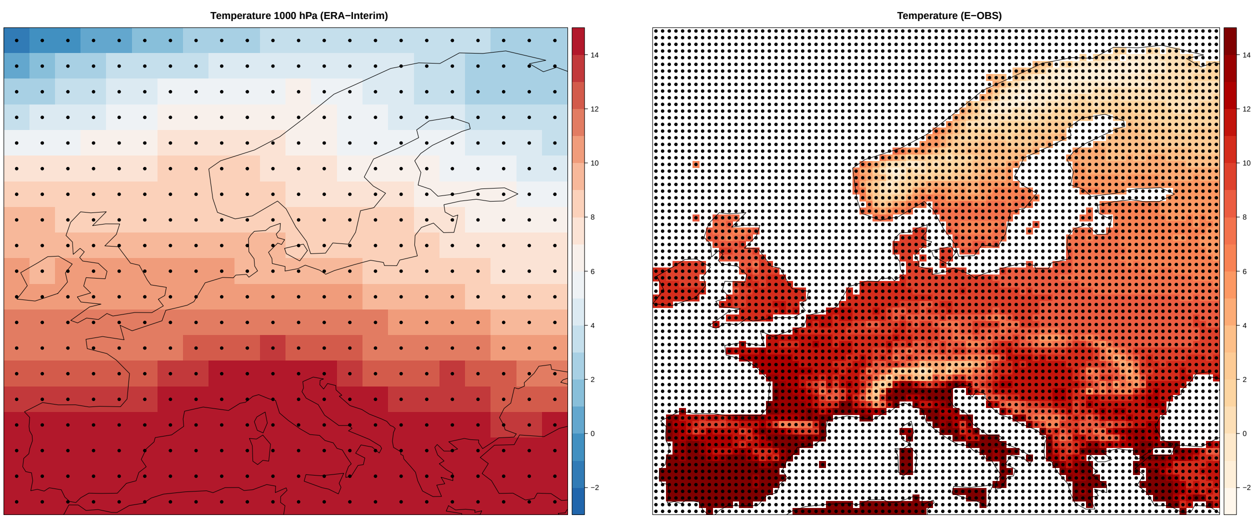
We can take a look at the grid resolutions of ERA-Interim and E-OBS in order to better visualize the gap we try to bridge with the downscaling.

```
cb <- colorRampPalette(brewer.pal(9, "OrRd"))(80)
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")
colsindex <- rev(brewer.pal(n = 9, "RdBu"))
```

```
cb2 <- colorRampPalette(colsindex)

pplot <- list()
pplot[[1]] <- spatialPlot(climatology(subsetGrid(xt,var = "ta@1000")), backdrop.theme = "coastline",
                          main = "Temperature 1000 hPa (ERA-Interim)",
                          col.regions = cb2,
                          at = seq(-3, 15, 1),
                          set.min = -3, set.max = 15, 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 = "Temperature (E-OBS)",
                          col.regions = cb,
                          at = seq(-3, 15, 1),
                          set.min = -3, set.max = 15, 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, or the percentiles 02th and 98th in order to gain knowledge about the observed data. To compute the statistics we use the library climate4R.value of climate4R.

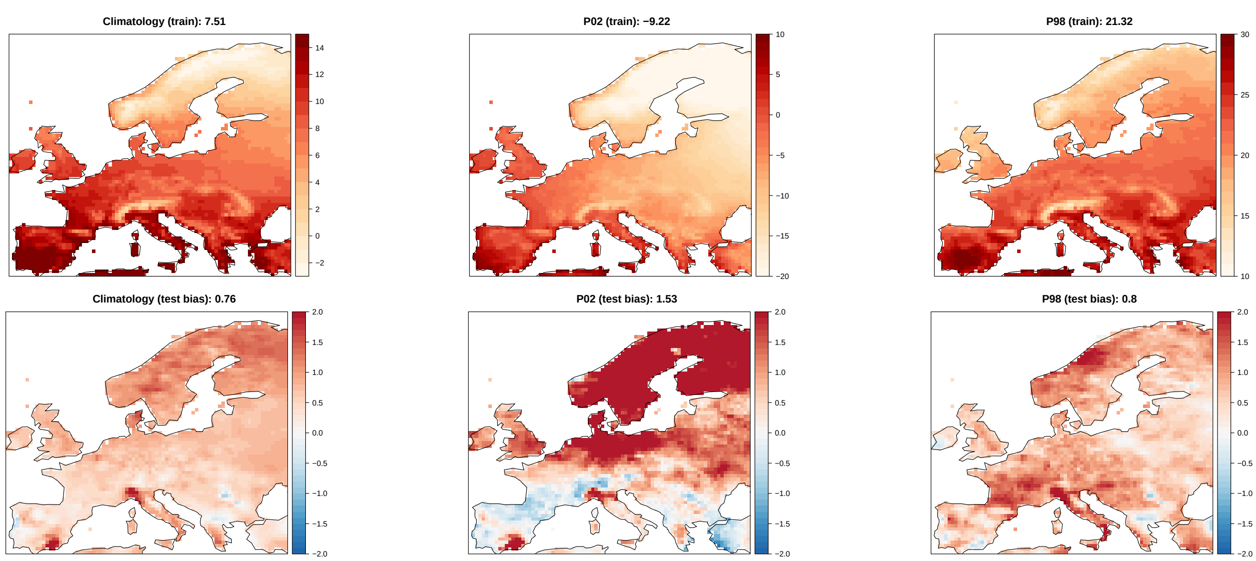
```
cb <- colorRampPalette(brewer.pal(9, "OrRd"))(80)
colsindex <- rev(brewer.pal(n = 9, "RdBu"))
cb2 <- colorRampPalette(colsindex)

pplot <- at <- list()
n1 <- 0; n2 <- 3
indexNames <- c("Climatology", "P02", "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(-3, 15, 1); at[[2]] <- seq(-2, 2, 0.1)
  }
  if (indexName == "P02") {
    indexTrain <- valueIndex(yT,index.code = "P02")$Index %>% redim()
    indexTest <- valueIndex(yt,index.code = "P02")$Index %>% redim()
    at[[1]] <- seq(-20, 10, 1); at[[2]] <- seq(-2, 2, 0.1)
  }
  if (indexName == "P98") {
    indexTrain <- valueIndex(yT,index.code = "P98")$Index %>% redim()
    indexTest <- valueIndex(yt,index.code = "P98")$Index %>% redim()
    at[[1]] <- seq(10, 30, 1); at[[2]] <- seq(-2, 2, 0.1)
  }

  for (i in 1:2) {
    if (i == 1) {
      dataset <- "(train)"; index <- indexTrain; n1 <- n1 + 1; n <- n1
      value <- index$Data; colorbar <- cb
    }
    if (i == 2) {
      indexTest <- gridArithmetics(indexTest,indexTrain,operator = "-")
      dataset <- "(test bias)"; index <- indexTest; n2 <- n2 + 1; n <- n2
      value <- abs(index$Data); colorbar <- cb2
    }
    pplot[[n]] <- spatialPlot(climatology(index), backdrop.theme = "coastline",
                              main = paste(indexName,paste0(dataset,":"),
                                           round(mean(value, na.rm = TRUE),digits = 2)),
                              col.regions = colorbar,
                              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)
```

Downscaling

Generalized Linear Models (GLM)

To downscale via generalized linear models (GLM) we rely on the downscaleR package of climate4R. In particular, we use the downscaleChunk function of downscaleR. In the case of temperature, the generalized linear model has a gaussian family with link identity which is, in fact, an ordinary least squares regression. Therefore, we input to the function the predictor (x), the number of local predictors to be used (neighbours), the predictand (y) and the test set where to apply the infered relationship as newdata. We save the predictions for loading of the data during validation. Note that downscaleChunk temporarily creates .rda files in your working directory, containing the predictions per chunk.

```
glmName <- c("glm1","glm4")
neighs <- c(1,4)
lapply(1:length(glmName), FUN = function(z) {
  pred <- downscaleChunk(x = xT, y = yT, newdata = list(xt),
                         method = "GLM", family = "gaussian", simulate = FALSE,
                         prepareData.args = list(local.predictors = list(n=neighs[z],
                                                                         vars = getVarNames(xT))))[[2]]
    redim(drop = TRUE)
  save(pred,file = paste0("./Data/temperature/predictions_",glmName[z],".rda"))
})
```

Downscaling - Deep Neural Networks

In the following code we define a function containing the deep learning topologies intercompared in the study.

```
deepName <- c("CNN-LM","CNN1","CNN10","CNN-PR","CNNdense")
architectures <- function(architecture,input_shape,output_shape) {
  if (architecture == "CNN-LM") {
    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)
    outputs = layer_dense(l4,units = output_shape)
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  if (architecture == "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)
    outputs = layer_dense(l4,units = output_shape)
```

```
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  if (architecture == "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)
    outputs = layer_dense(l4,units = output_shape)
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  if (architecture == "CNN-PR") {
    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)
    outputs = layer_dense(l4,units = output_shape)
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  if (architecture == "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")
    outputs = layer_dense(l6,units = output_shape)
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  return(model)
}
```

We prepare the predictor and predictand datasets for integration with keras with the functions prepareData.keras and prepareNewData.keras

```
xy.T <- prepareData.keras(xT,yT,
                          first.connection = "conv",
                          last.connection = "dense",
                          channels = "last")
xy.t <- prepareNewData.keras(xt,xy.T)
```

We loop over the topologies to train the deep models and predict over the test set. Unlike GLMs where there was a model per gridpoint, deep models perform multi-task, downscaling to all sites at a time.

```
lapply(1:length(deepName), FUN = function(z){
  model <- architectures(architecture = deepName[z],
```

```
                         input_shape = dim(xy.T$x.global)[-1],
                         output_shape = dim(xy.T$y$Data)[2])
  downscaleTrain.keras(obj = xy.T,
                       model = model,
                       clear.session = TRUE,
                       compile.args = list("loss" = "mse",
                                           "optimizer" = optimizer_adam(lr = 0.0001)),
                       fit.args = list("batch_size" = 100,
                                       "epochs" = 10,
                                       "validation_split" = 0.1,
                                       "verbose" = 1,
                                       "callbacks" = list(callback_early_stopping(patience = 30),
                                                 callback_model_checkpoint(
                                                 filepath=paste0('./models/temperature/',deepName[z],'.h
                                                                  monitor='val_loss', save_best_only=TRU
  pred <- downscalePredict.keras(newdata = xy.t,
                                 model = list("filepath" =
                                                paste0("./models/temperature/",deepName[z],".h5")),
                                 C4R.template = yT,
                                 clear.session = TRUE)
  save(pred,file = paste0("./Data/temperature/predictions_",deepName[z],".rda"))
})
```

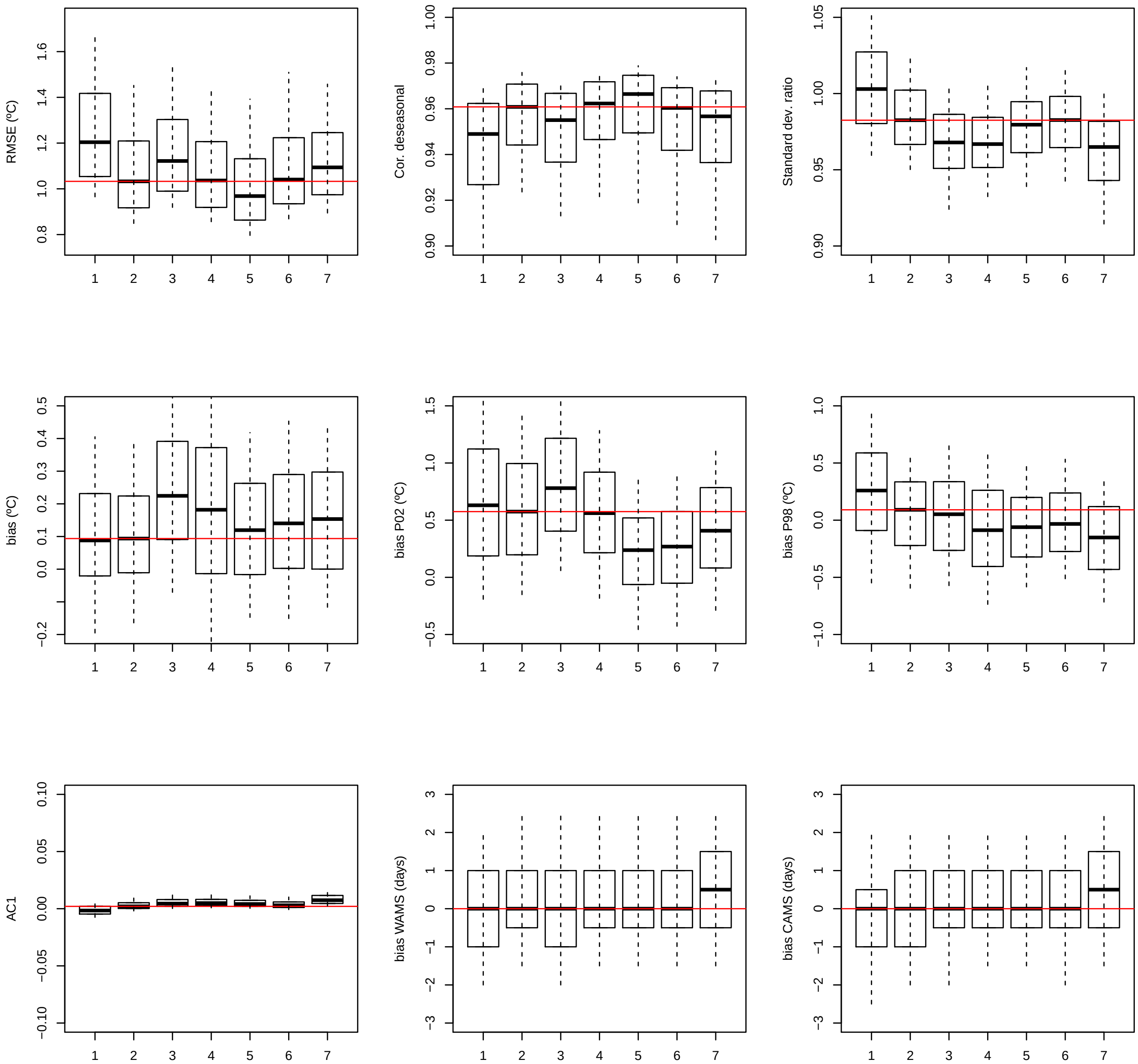
Validation of the Results

In this code, we calculate the validation indices by using the library climate4R.value of climate4R. In particular the indices used are: the Root Mean Squared Error (RMSE), the deseasonal perason correlation, the biases of the climatology and of the percentile 2th and 98th and the ratio of the standard deviations.

```
models <- c("glm1","glm4",
            "CNN-LM","CNN1","CNN10",
            "CNN-PR","CNNdense")
measures <- c("ts.RMSE","ts.rp","ratio",rep("bias",6))
index <- c(rep(NA,2),"sd","Mean","P02","P98","AC1",
           "WarmAnnualMaxSpell","ColdAnnualMaxSpell")
yt2 <- scaleGrid(yt,time.frame = "daily",window.width = 31) %>% redim(drop=TRUE)
validation.list <- lapply(1:length(measures), FUN = function(z) {
  lapply(1:length(models), FUN = function(zz){
    args <- list()
    load(paste0("./Data/temperature/predictions_",models[zz],".rda"))
    if (any(measures[z] == c("ts.rp","ratio"))) {
      pred2 <- scaleGrid(pred,time.frame = "daily",window.width = 31) %>% redim(drop=TRUE)
      args[["y"]] <- yt2; args[["x"]] <- pred2
    } else {
      args[["y"]] <- yt; args[["x"]] <- pred
    }
    args[["measure.code"]] <- measures[z]
    if (!is.na(index[z])) args[["index.code"]] <- index[z]
    do.call("valueMeasure",args)$Measure
  }) %>% makeMultiGrid()
})
save(validation.list, file = "./Data/temperature/validation.rda")
```

Once the validation indices are calculated, we represent the results in boxplots.

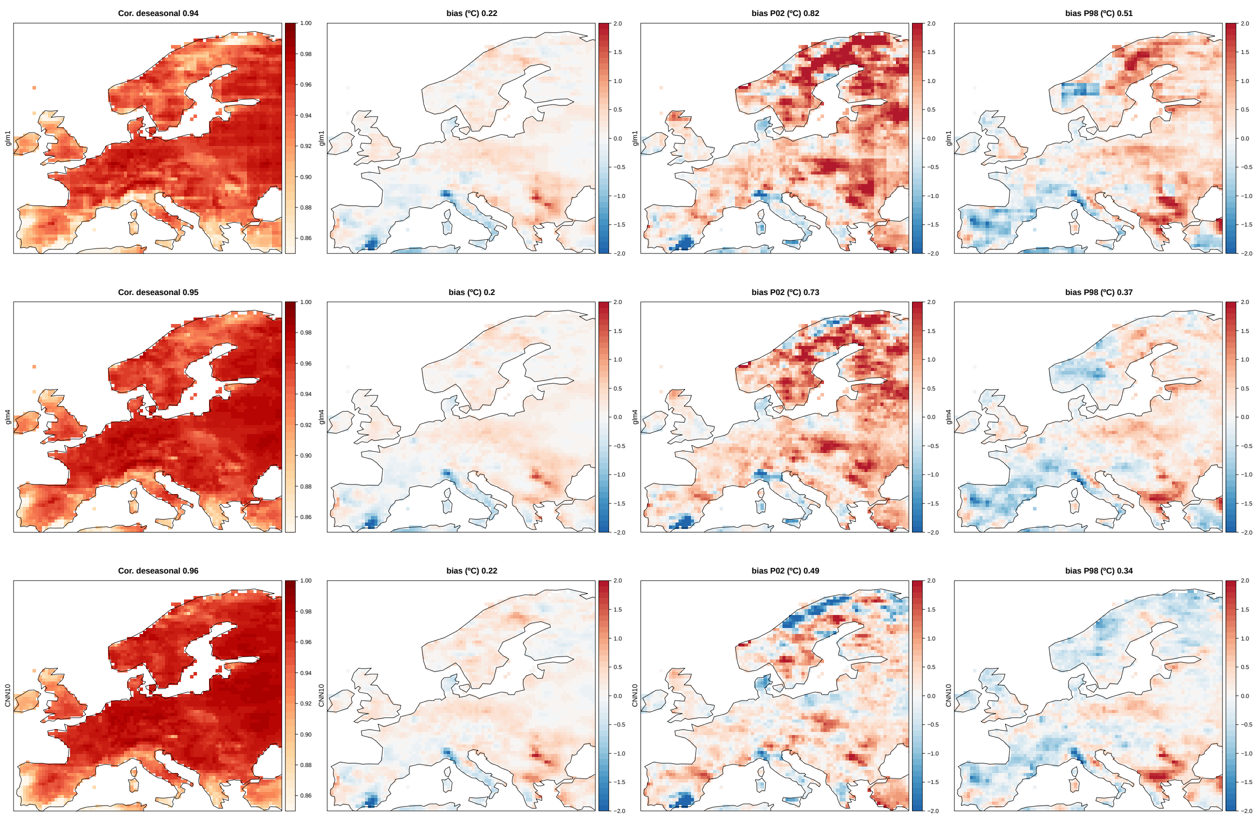
```
ylabs <- c("RMSE (ºC)","Cor. deseasonal",
           "Standard dev. ratio","bias (ºC)",
           "bias P02 (ºC)","bias P98 (ºC)",
           "AC1", "bias WAMS (days)",
           "bias CAMS (days)")
par(mfrow = c(3,3))
lapply(1:length(validation.list), FUN = function(z) {
  if (z == 1) {ylim <- c(0.75,1.75)}
  if (z == 2) {ylim <- c(0.9,1)}
  if (z == 3) {ylim <- c(0.9,1.05)}
  if (z == 4) {ylim <- c(-0.2,0.5)}
  if (z == 5) {ylim <- c(-0.5,1.5)}
  if (z == 6) {ylim <- c(-1,1)}
  if (z == 7) {ylim <- c(-0.1,0.1)}
  if (any(z == c(8,9))) {ylim <- c(-3,3)}
  index <- (validation.list[[z]] %>% redim(drop = TRUE))$Data
  dim(index) <- c(nrow(index),prod(dim(index)[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, ylim = ylim, range = 0.0001, ylab = ylabs[z], asp = 1)
  lines(c(0,8),c(mglm4,mglm4), col = "red")
  for (i in 1:ncol(index)) lines(c(i,i),perc[,i], lty = 2)
})
```



In order to obtain a spatial representation we use the function spatialPlot of visualizeR for the

```
ylabs <- c("glm1","glm4",NA,NA,"CNN10")
mains <- c(NA,"Cor. deseasonal",NA,"bias (ºC)","bias P02 (ºC)","bias P98 (ºC)")
cb <- colorRampPalette(brewer.pal(9, "OrRd"))(80)
colsindex <- rev(brewer.pal(n = 9, "RdBu"))
cb2 <- colorRampPalette(colsindex)
validation.plots <- lapply(c(2,4,5,6),FUN = function(z) {
  lapply(c(1,2,5),FUN = function(zz) {
    if (z == 2) {
      at <- seq(0.85, 1, 0.005); colorbar <- cb
    } else {
      at <- seq(-2, 2, 0.1); colorbar <- cb2
    }
    index <- subsetDimension(validation.list[[z]],dimension = "var",indices = zz) %>%
      redim(drop = TRUE)
    spatialPlot(index, backdrop.theme = "coastline",
```

```
                  ylab = ylabs[zz],
                  main = paste(mains[z],
                               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 = cbind(1:3,4:6,7:9,10:12)
grid.arrange(grobs = unlist(validation.plots,recursive = FALSE), layout_matrix = lay)
```



Precipitation

In this section we present the code needed to downscale precipitation. Though the steps taken are very similar to those of temperature there are some particularities that are good to mention. We start by loading the precipitation using 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) and use binaryGrid to convert every value greater than 1 equal to 1 and lower or equal than 1 equal to 0.

```
# Train
yT <- subsetGrid(y,years = 1979:2002)
yT_bin <- binaryGrid(yT,threshold = 1,condition = "GT")
```

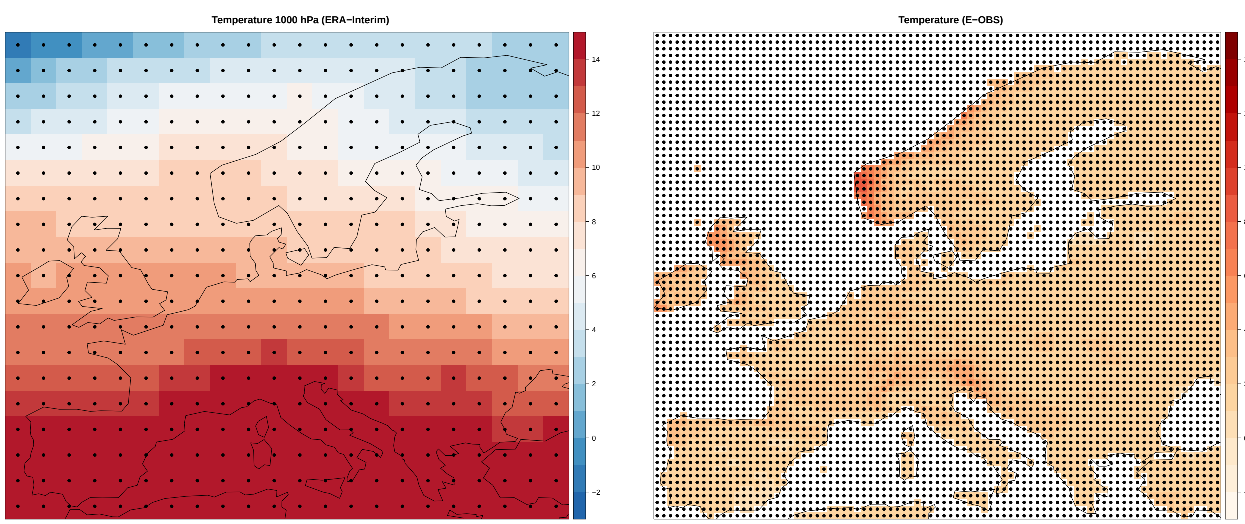
```
# Test
yt <- subsetGrid(y,years = 2003:2008)
yt_bin <- binaryGrid(yt,threshold = 1,condition = "GT")
```

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.

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

pplot <- list()
pplot[[1]] <- spatialPlot(climatology(subsetGrid(xt,var = "ta@1000")), backdrop.theme = "coastline",
                          main = "Temperature 1000 hPa (ERA-Interim)",
                          col.regions = cb2,
                          at = seq(-3, 15, 1),
                          set.min = -3, set.max = 15, 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 = "Temperature (E-OBS)",
                          col.regions = cb,
                          at = seq(-3, 15, 1),
                          set.min = -3, set.max = 15, 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)
```



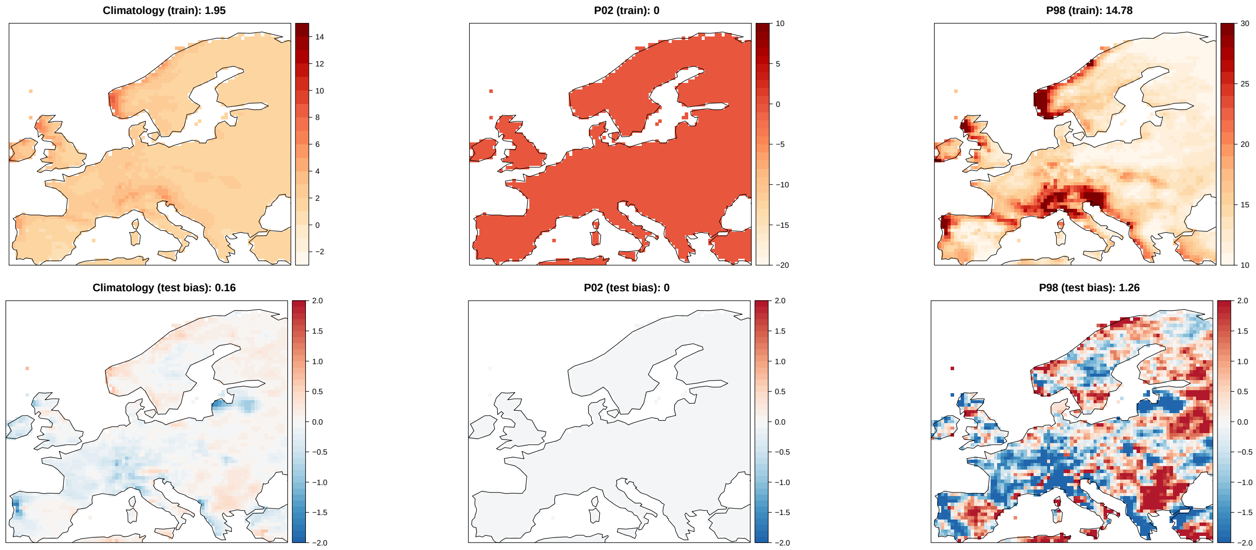
We can visualize some statistics of the train and test distributions, such as the climatology, or the percentiles 02th and 98th in order to gain knowledge about the observed data. To compute the statistics we use the library climate4R.value of climate4R.

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

pplot <- at <- list()
n1 <- 0; n2 <- 3
indexNames <- c("Climatology", "P02", "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(-3, 15, 1); at[[2]] <- seq(-2, 2, 0.1)
  }
  if (indexName == "P02") {
    indexTrain <- valueIndex(yT,index.code = "P02")$Index %>% redim()
    indexTest <- valueIndex(yt,index.code = "P02")$Index %>% redim()
    at[[1]] <- seq(-20, 10, 1); at[[2]] <- seq(-2, 2, 0.1)
  }
  if (indexName == "P98") {
    indexTrain <- valueIndex(yT,index.code = "P98")$Index %>% redim()
    indexTest <- valueIndex(yt,index.code = "P98")$Index %>% redim()
    at[[1]] <- seq(10, 30, 1); at[[2]] <- seq(-2, 2, 0.1)
  }

  for (i in 1:2) {
    if (i == 1) {
      dataset <- "(train)"; index <- indexTrain; n1 <- n1 + 1; n <- n1
      value <- index$Data; colorbar <- cb
    }
    if (i == 2) {
      indexTest <- gridArithmetics(indexTest,indexTrain,operator = "-")
      dataset <- "(test bias)"; index <- indexTest; n2 <- n2 + 1; n <- n2
      value <- abs(index$Data); colorbar <- cb2
    }
    pplot[[n]] <- spatialPlot(climatology(index), backdrop.theme = "coastline",
                          main = paste(indexName,paste0(dataset,":"),
                                       round(mean(value, na.rm = TRUE),digits = 2)),
                          col.regions = colorbar,
                          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)
```



Downscaling

Generalized Linear Models (GLM)

To downscale via generalized linear models (GLM) we rely on the downscaleR package of climate4R. In particular, we use the downscaleChunk function of downscaleR. In the case of precipitation, there are 2 generalized linear models: one to predict the occurrence of precipitation with binomial family and link logit and another to predict tha rainfall amount based on a gamma family and link logarithmic. Therefore, we input to the function the predictor (x), the number of local predictors to be used (neighbours), the predictand (y) and the test set where to apply the infered relationship as newdata. We save the predictions for loading of the data during validation. Note that downscaleChunk temporarily creates .rda files in your working directory, containing the predictions per chunk.

```
simulateName <- c("deterministic","stochastic")
glmName <- c("glm1","glm4")
neighs <- c(1,4)
y.ocu <- binaryGrid(yT,condition = "GT",threshold = 1)
y.rest <- gridArithmetics(yT,1,operator = "-")
simulateGLM <- c(FALSE,TRUE)
lapply(1:length(glmName), FUN = function(z){
  lapply(1:length(simulateGLM),FUN = function(zz) {
    pred <- downscaleChunk(x = xT, y = y.ocu, newdata = list(xt),
                           method = "GLM",
                           family = binomial(link = "logit"),
                           simulate = simulateGLM[zz],
                           prepareData.args = list(local.predictors = list(n=neighs[z],
                                                                         vars = getVarNames(xT)))
    )
    pred_ocu_train <- pred[[1]] %>% redim(drop = TRUE)
    pred_ocu <- pred[[2]] %>% redim(drop = TRUE)
    rm(pred)
    pred_amo <- downscaleChunk(x = xT, y = y.rest, newdata = list(xt),
                               method = "GLM",
                               family = Gamma(link = "log"),
                               simulate = simulateGLM[zz],
```

```
                                condition = "GT", threshold = 0,
                                prepareData.args = list(local.predictors = list(n=neighs[z],
                                                                                vars = getVarNames(xT)))
      gridArithmetics(1,operator = "+") %>% redim(drop = TRUE)
    pred_bin <- binaryGrid(pred_ocu,ref.obs = yT_bin,ref.pred = pred_ocu_train); rm(pred_ocu_train)
    save(pred_bin,pred_ocu,pred_amo,
         file = paste0("./Data/precip/predictions_",simulateName[zz],"_",glmName[z],".rda"))
  })
})
```

Downscaling - Deep Neural Networks

Equal to temperature we define a function containing the topologies intercompared in the study. The main difference in comparison with downscaling temperature is that for precipitation we infer the probability of rain and the shape and scale parameters of a Gamma distribution, and therefore there are three output parameters per predictand's gridpoint.

```
deepName <- c("CNN-LM","CNN1","CNN10","CNN-PR","CNNdense")
architectures <- function(architecture,input_shape,output_shape) {
  if (architecture == "CNN-LM") {
    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 (architecture == "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 (architecture == "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 (architecture == "CNN-PR") {
    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 (architecture == "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)
}
```

We prepare the predictor and predictand datasets. We substract 1 to the precipitation (yT) to center the conditional Gamma distribution in 0. This will be later added to the prediction output.

```
xy.T <- prepareData.keras(xT,gridArithmetics(yT,1,operator = "-"),
                          first.connection = "conv",
                          last.connection = "dense",
                          channels = "last")
xy.tT <- prepareNewData.keras(xT,xy.T)
xy.t <- prepareNewData.keras(xt,xy.T)
```

We loop over the topologies to train the deep models and predict over the test set. Unlike GLMs where there was two models, one for the occurrence of rain and other for the amount of rain, with deep learning we minimize the negative log-likelihood of a Bernouilli Gamma distribution and therefore both the occurrence and quantity of rain for a given day are derived from these infered parameters. The custom loss function 'bernouilliGamma.loss_function' is part of the downscaleR.keras. In addition we use the downscaleR.keras function 'bernouilliGamma.statistics' to compute the deterministic (i.e., expectance of the conditional distribution) or the stochastic (i.e., sample from the conditional distribution) prediction.

```
simulateName <- c("deterministic","stochastic")
simulateDeep <- c(FALSE,TRUE)
lapply(1:length(deepName), FUN = function(z){
  model <- architectures(architecture = deepName[z],
                         input_shape = dim(xy.T$x.global)[-1],
                         output_shape = dim(xy.T$y$Data)[2])
  downscaleTrain.keras(obj = xy.T,
             model = model,
             clear.session = TRUE,
             compile.args = list("loss" = bernouilliGamma.loss_function(last.connection = "dense"),
                                 "optimizer" = optimizer_adam(lr = 0.0001)),
                       fit.args = list("batch_size" = 100,
                            "epochs" = 1000,
                            "validation_split" = 0.1,
                            "verbose" = 1,
                            "callbacks" = list(callback_early_stopping(patience = 30),
                                callback_model_checkpoint(filepath=paste0('./models/precip/',deepName[z
                                monitor='val_loss', save_best_only=TRUE))))
  lapply(1:length(simulateDeep),FUN = function(zz) {
    pred_ocu_train <- downscalePredict.keras(newdata = xy.tT,
                                            model = list("filepath" =
                                                     paste0("./models/precip/",deepName[z],".h5"),
                                                    "custom_objects" =
                                                    c("custom_loss" =
                                                     bernouilliGamma.loss_function(
                                                       last.connection = "dense"))),
                                            C4R.template = yT,
                                            clear.session = TRUE) %>%
      subsetDimension(dimension = "var", indices = 1)
    pred <- downscalePredict.keras(newdata = xy.t,
                                  model = list("filepath" =
                                              paste0("./models/precip/",deepName[z],".h5"),
                                              "custom_objects" =
                                              c("custom_loss" =
                                              bernouilliGamma.loss_function(last.connection = "dense"))
                                  C4R.template = yT,
                                  clear.session = TRUE)
    pred <- bernouilliGamma.statistics(p = subsetDimension(pred, dimension = "var", indices = 1),
                                      alpha = subsetDimension(pred, dimension = "var", indices = 2),
                                      beta = subsetDimension(pred, dimension = "var", indices = 3),
                                      simulate = simulateDeep[zz],
                                      bias = 1)
    pred_ocu <- subsetDimension(pred,dimension = "var",indices = 1) %>% redim(drop = TRUE)
    pred_amo <- subsetDimension(pred,dimension = "var",indices = 2) %>% redim(drop = TRUE)
    pred_bin <- binaryGrid(pred_ocu,ref.obs = yT_bin,ref.pred = pred_ocu_train); rm(pred_ocu_train)
    save(pred_bin,pred_ocu,pred_amo,file =
            paste0("./Data/precip/predictions_",simulateName[zz],"_",deepName[z],".rda"))
  })
})
```

Validation of the Results

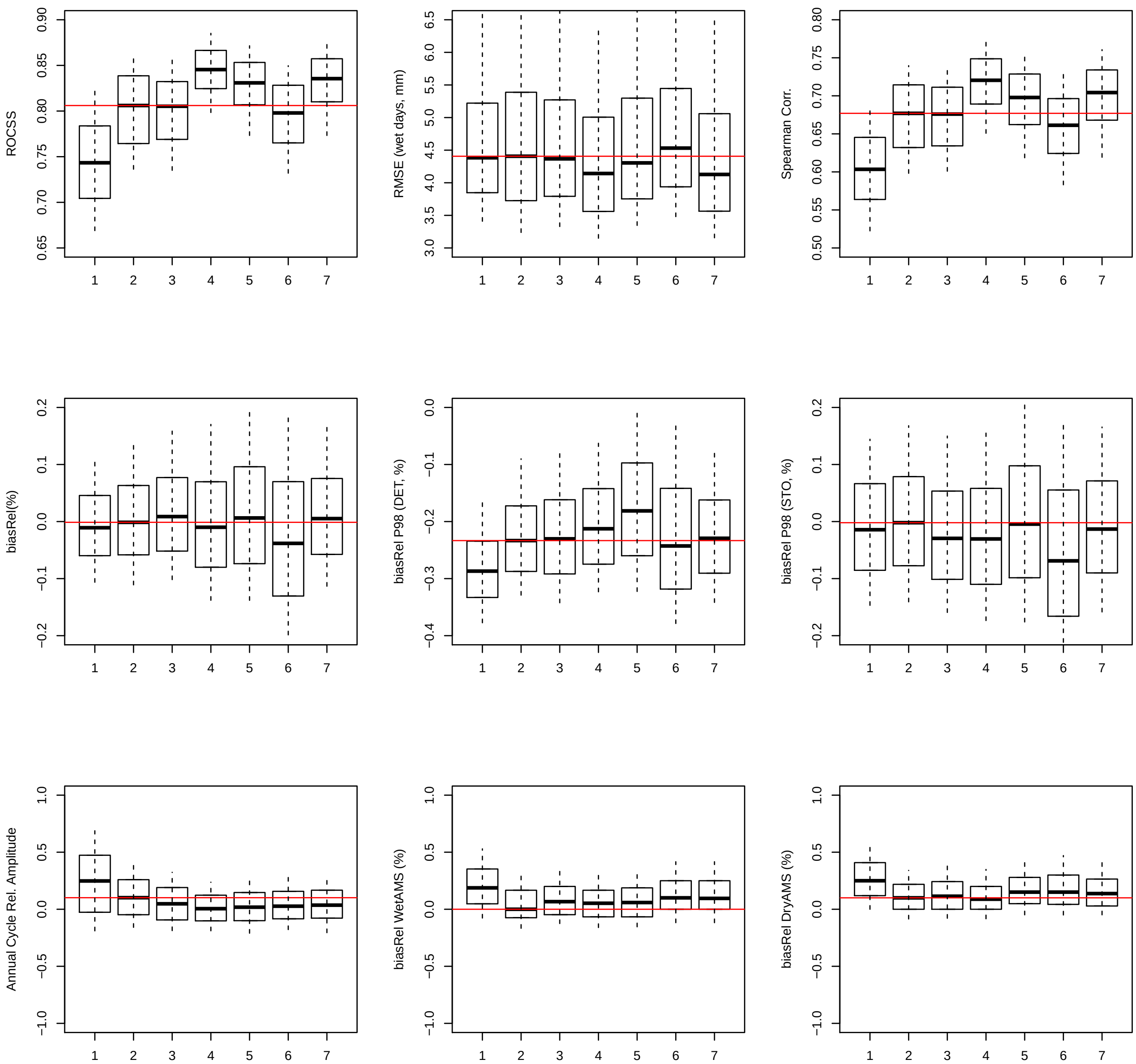
In this code, we calculate the validation indices by using the library climate4R.value of climate4R. In particular the indices used are: the Root Mean Squared Error (RMSE), the deseasonal perason correlation, the biases of the climatology and of the percentile 2th and 98th and the ratio of the standard deviations.

```
simulateName <- c(rep("deterministic",5),"stochastic",rep("deterministic",3))
models <- c("glm1","glm4",
               "CNN-LM","CNN1","CNN10",
               "CNN-PR","CNNdense")
measures <- c("ts.rocss","ts.RMSE","ts.rs",rep("biasRel",6))
index <- c(rep(NA,3),"Mean",rep("P98",2),"AnnualCycleRelAmp",
           "WetAnnualMaxSpell","DryAnnualMaxSpell")
validation.list <- lapply(1:length(measures), FUN = function(z) {
  lapply(1:length(models), FUN = function(zz){
    args <- list()
    load(paste0("./Data/precip/predictions_",simulateName[z],"_",models[zz],".rda"))
    if (simulateName[z] == "deterministic") {
      pred <- gridArithmetics(pred_bin,pred_amo,operator = "*")
      if (measures[z] == "ts.rocss") {
        args[["y"]] <- yt_bin; args[["x"]] <- pred_ocu
      } else if (measures[z] == "ts.RMSE") {
        args[["y"]] <- yt; args[["x"]] <- pred_amo
        args[["condition"]] = "GT"; args[["threshold"]] = 1; args[["which.wetdays"]] = "Observation"
      } else {
        args[["y"]] <- yt; args[["x"]] <- pred
      }
    } else {
      pred <- gridArithmetics(pred_ocu,pred_amo,operator = "*")
      args[["y"]] <- yt; args[["x"]] <- pred
    }
    args[["measure.code"]] <- measures[z]
    if (!is.na(index[z])) args[["index.code"]] <- index[z]
    do.call("valueMeasure",args)$Measure
  }) %>% makeMultiGrid()
})
save(validation.list, file = "./Data/precip/validation.rda")
```

Once the validation indices are calculated, we represent the results in boxplots.

```
par(mfrow = c(3,3)) #
ylabs <- c("ROCSS","RMSE (wet days, mm)",
           "Spearman Corr.","biasRel(%)",
           "biasRel P98 (DET, %)","biasRel P98 (STO, %)",
           "Annual Cycle Rel. Amplitude", "biasRel WetAMS (%)",
           "biasRel DryAMS (%)")
lapply(1:length(validation.list), FUN = function(z) {
  if (z == 1) {ylim <- c(0.65,0.9)}
  if (z == 2) {ylim <- c(3,6.5)}
  if (z == 3) {ylim <- c(0.5,0.8)}
  if (z == 4) {ylim <- c(-0.2,0.2)}
  if (z == 5) {ylim <- c(-0.4,0.0)}
  if (z == 6) {ylim <- c(-0.2,0.2)}
  if (z == 7) {ylim <- c(-1,1)}
  if (any(z == c(8,9))) {ylim <- c(-1,1)}
```

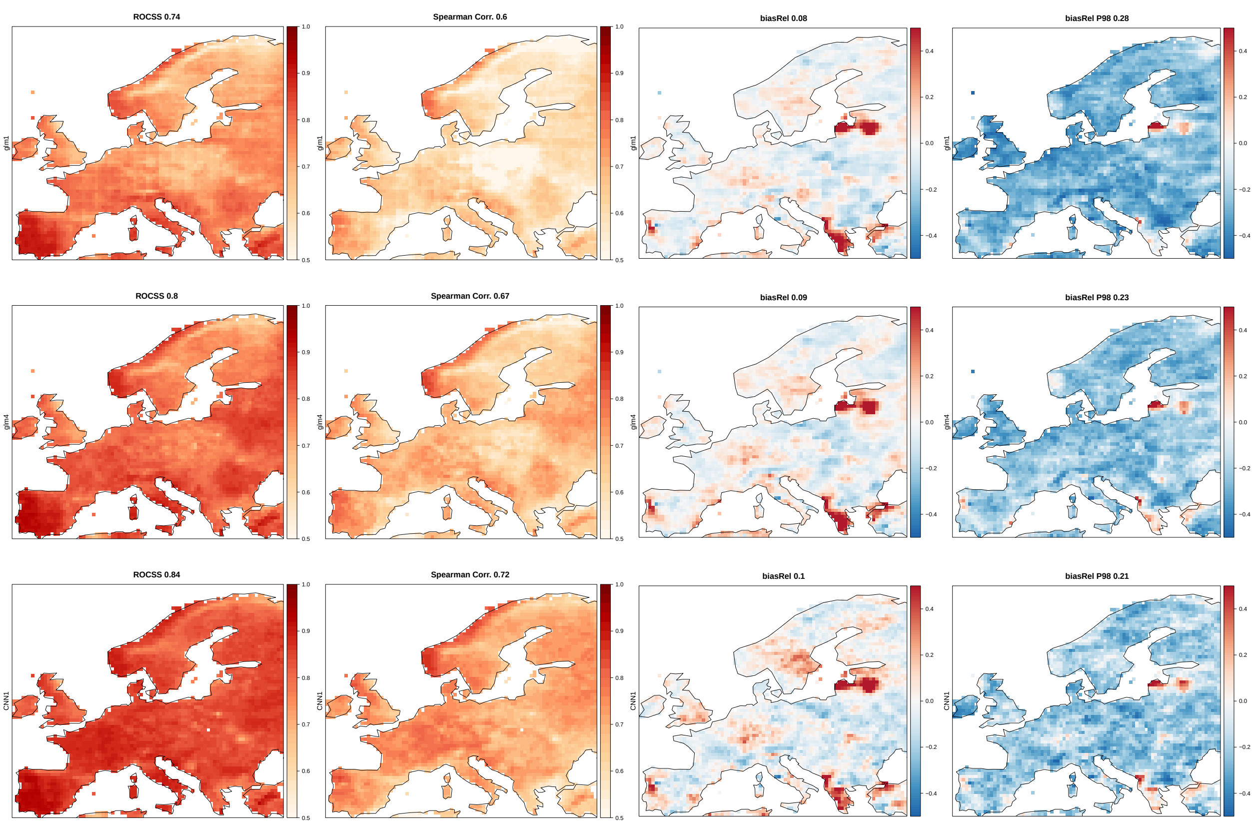
```
  index <- (validation.list[[z]] %>% redim(drop = TRUE))$Data
  dim(index) <- c(nrow(index),prod(dim(index)[2:3]))
  indLand <- (!apply(index,MARGIN = 2,anyNA)) %>% which()
  index <- index[,indLand] %>% t()
  mglm4 <- median(index[,2],na.rm = TRUE)
  perc <- apply(index,MARGIN = 2,FUN = function(z) quantile(z,probs = c(0.1,0.9)))
  boxplot(index, outline = FALSE, ylim = ylim, range = 0.0001, ylab = ylabs[z], asp = 1)
  lines(c(0,8),c(mglm4,mglm4), col = "red")
  for (i in 1:ncol(index)) lines(c(i,i),perc[,i], lty = 2)
})
```



In order to obtain a spatial representation we use the function spatialPlot of visualizeR for the

```
ylabs <- c("glm1","glm4",NA,"CNN1")
mains <- c("ROCSS",NA,"Spearman Corr.","biasRel","biasRel P98")
cb <- colorRampPalette(brewer.pal(9, "OrRd"))(80)
colsindex <- rev(brewer.pal(n = 9, "RdBu"))
```

```r
cb2 <- colorRampPalette(colsindex)
validation.plots <- lapply(c(1,3,4,5),FUN = function(z) {
  lapply(c(1,2,4),FUN = function(zz) {
    if (z == 1) {
      at <- seq(0.5, 1, 0.01); colorbar <- cb
    } else if (z == 3) {
      at <- seq(0.5, 1, 0.02); colorbar <- cb
    } else {
      at <- seq(-0.5, 0.5, 0.01); colorbar <- cb2
    }
    index <- subsetDimension(validation.list[[z]],dimension = "var",indices = zz) %>% redim(drop = TRUE)
    spatialPlot(index, backdrop.theme = "coastline",
                ylab = ylabs[zz],
                main = paste(mains[z],
                             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 = cbind(1:3,4:6,7:9,10:12)
grid.arrange(grobs = unlist(validation.plots,recursive = FALSE), layout_matrix = lay)
```



Technical Aspects

To perform all the stages involved in this study we relied on the local machine described below.

1. Local Machine (HP-ProDesk-600-G2-MT)
    - Operating system: ubuntu 16.04 LTS
    - Memory: 15.6 GiB
    - Processor: Intel® Core™ i7-6700 CPU @ 3.40GHz × 8
    - SO: 64 bits
    - Disc: 235.1 GiB