

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

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

*2019-07-03*

## Contents

<b>1</b>	<b>Loading Data</b>	<b>1</b>
<b>2</b>	<b>Downscaling</b>	<b>4</b>
2.1	Generalized Linear Models (GLM)	4
2.2	Downscaling - Deep Neural Networks	5
<b>3</b>	<b>Validation of the Results</b>	<b>8</b>
3.1	Intercomparison of Methods	8
3.2	Spatial Maps	11

## 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 = "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)

save(xT,file = "./Data/temperature/xT.rda")
save(xt,file = "./Data/temperature/xt.rda")
save(yT,file = "./Data/temperature/yT.rda")
save(yt,file = "./Data/temperature/yt.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.

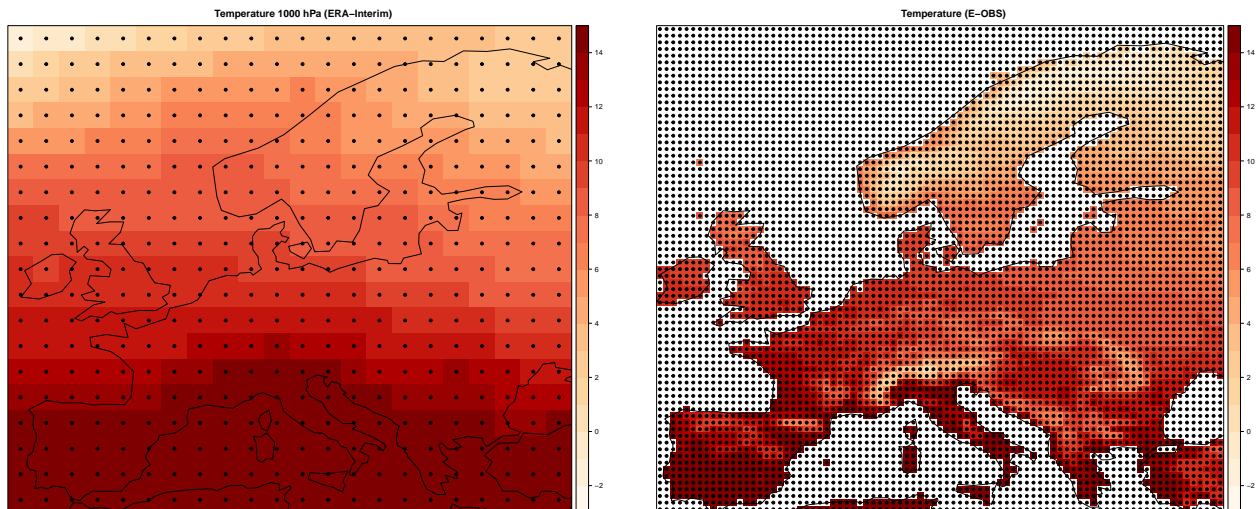
```

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")

pplot <- list()
pplot[[1]] <- spatialPlot(climatology(subsetGrid(xt,var = "ta@1000")), backdrop.theme = "coastline",
  main = "Temperature 1000 hPa (ERA-Interim)",
  col.regions = cb,
  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, 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`.

```
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)
  }
}

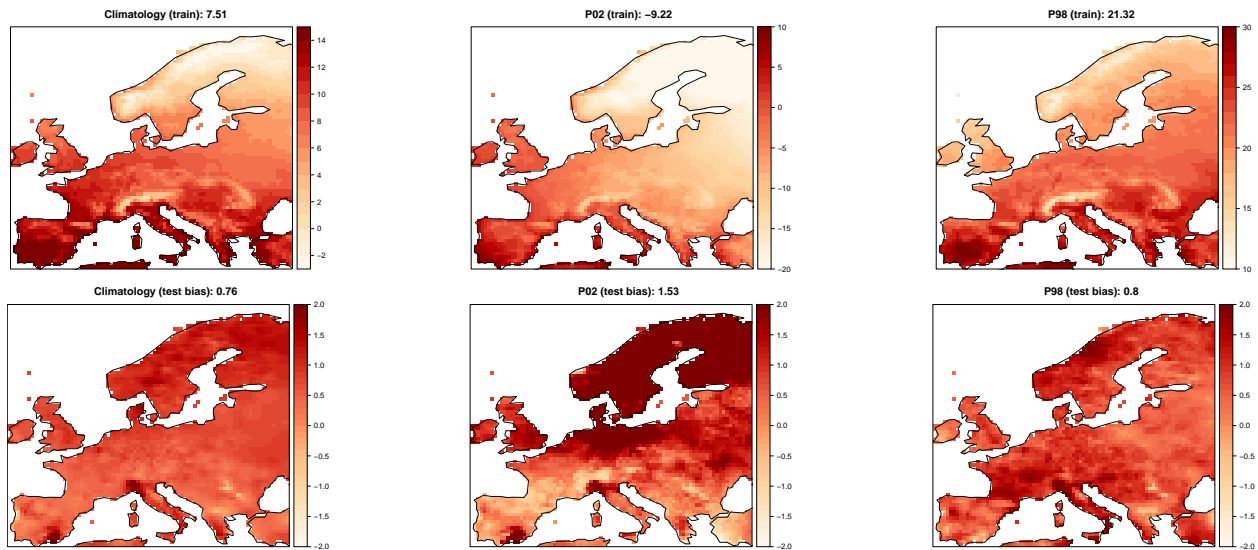
for (i in 1:2) {
  if (i == 1) {
    dataset <- "(train)"; index <- indexTrain; n1 <- n1 + 1; n <- n1
    value <- index$Data; colorbar <- cb
  }
  if (i == 2) {
    indexTest <- gridArithmetics(indexTest, indexTrain, operator = "-")
    dataset <- "(test bias)"; index <- indexTest; n2 <- n2 + 1; n <- n2
    value <- abs(index$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 = 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 `transformerR`.

```

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. 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.

```

trainPredictGLM <- function(x,y,newdata,neighbours=1,filename) {
  downscaleChunk(x = x, y = y, newdata = newdata,
    method = "GLM", family = "gaussian", simulate = "no",
    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)))
}

```

```

pred_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)))
pred_test <- bindGrid(chunk.list, dimension = "lat")
file.remove(lf)

pred <- pred_train
save(pred,file = paste0("./Data/temperature/predictions_train_",filename,".rda"))
pred <- pred_test
save(pred,file = paste0("./Data/temperature/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 the following we provide the code related to the deep learning architectures.

```

arquitectures <- function(arquitecture,input_shape,output_shape) {
  if (arquitecture == "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)
    outputs = layer_dense(l4,units = output_shape)
    model <- keras_model(inputs = inputs, outputs = outputs)
  }

  if (arquitecture == "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 (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)
  outputs = layer_dense(l4,units = output_shape)
  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)
  outputs = layer_dense(l4,units = output_shape)
  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")
  outputs = layer_dense(l6,units = output_shape)
  model <- keras_model(inputs = inputs, outputs = outputs)
}

return(model)
}

```

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

```

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

  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/temperature/',arquitechture,'.h

```

```

                                monitor='val_loss', save_best_only=TRUE)
)

model <- arquitectures(arquitecture,input_shape = dim(x)[-1],output_shape = ncol(y))
model %>% compile(optimizer = optimizer_adam(lr = learning_rate), loss = "mse")
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,arquitecture = "CNN1lin")
trainDEEP(xT,yT,arquitecture = "CNN1")
trainDEEP(xT,yT,arquitecture = "CNN10")
trainDEEP(xT,yT,arquitecture = "CNNinverse")
trainDEEP(xT,yT,arquitecture = "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, according to the loss function, the net estimates the parameters  $p$ ,  $\alpha$  and  $\beta$  of a Bernoulli-Gamma distribution. In the deterministic way, the prediction for a given day is the expectance of the conditional Bernoulli-Gamma distribution inferred. In the stochastic way, we sample from the conditional Bernoulli-Gamma distribution. This sampling is coded in the functions simulateOcu and simulateReg.

```

predictDEEP <- function(x,template,arquitecture,dataset) {
  model <- load_model_hdf5(filepath = paste0("./models/temperature/",arquitecture,".h5"))
  x <- x$Data %>% aperm(c(2,3,4,1))
  pred <- model$predict(x)
  ntime <- dim(template$Data)[1]
  nlat <- dim(template$Data)[2]
  nlon <- dim(template$Data)[3]

  # 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)
    out <- template
    out$Data[,indLand] <- grid
    dim(out$Data) <- c(ntime,nlat,nlon)
    return(out)
  }
  pred <- convert2map2D(pred,template)

  save(pred,
        file = paste0("./Data/temperature/predictions_",dataset,"_",arquitecture,".rda"))
  k_clear_session()
}

```



Then we predict and save the predictions.

```
# 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`.

```
# file_name <- paste0("./figures/fig06_temperature.pdf")
# pdf(file = file_name,width = 5,height = 10)
par(mfrow = c(3,2))

yt2 <- scaleGrid(yt,time.frame = "daily",window.width = 31) %>% redim(drop=TRUE)
filepreds=list("glm1","glm4",
               "CNN1lin","CNN1","CNN10",
               "CNNinverse","CNNdense")
datasetspred <- length(filepreds)
indexNames <- c("RMSE","Cor. deseasonal",
               "bias","Stand dev.", "bias02","bias98")
pplot <- list(); n <- 0
for (indexName in indexNames) {
  index <- array(dim = c(datasetspred,dim(yt$Data)[2:3]))
  for (i in 1:datasetspred) {
    load(paste0("./Data/temperature/predictions_test_",filepreds[[i]],".rda"))

    if (indexName == "RMSE") {
      index[i,,] <- valueMeasure(yt,pred,measure.code="ts.RMSE")$Measure$Data
      ylim <- c(0.75,1.75)
    }

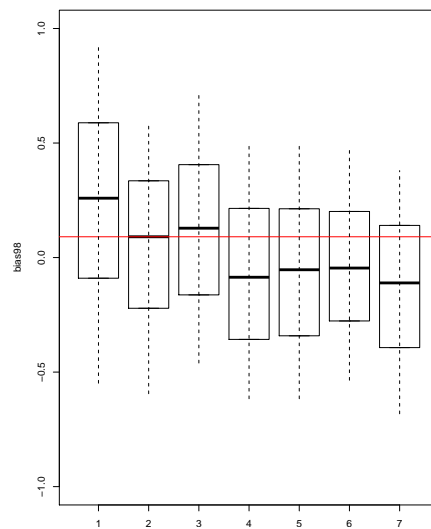
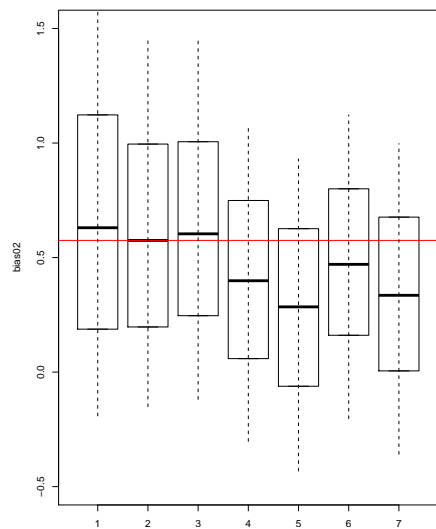
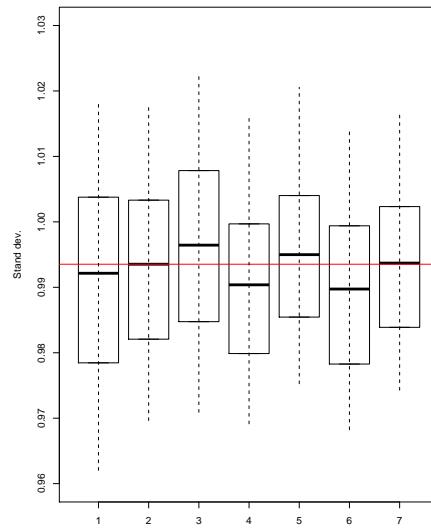
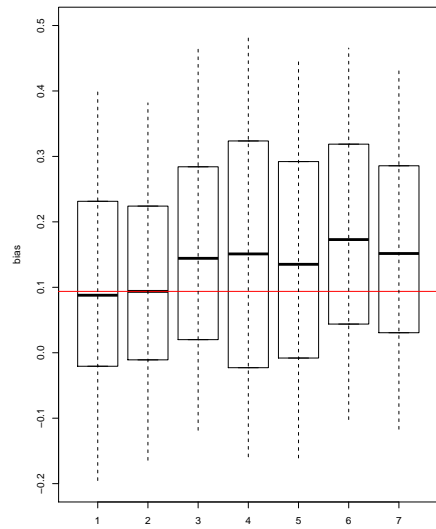
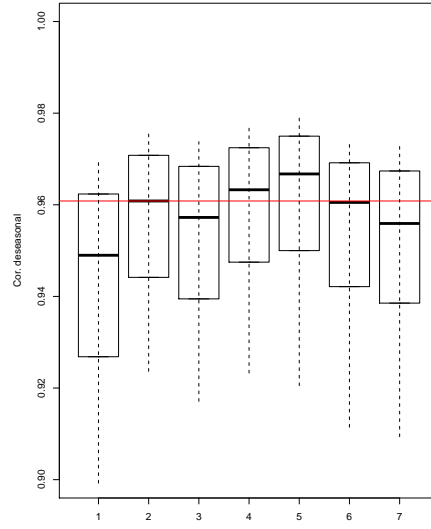
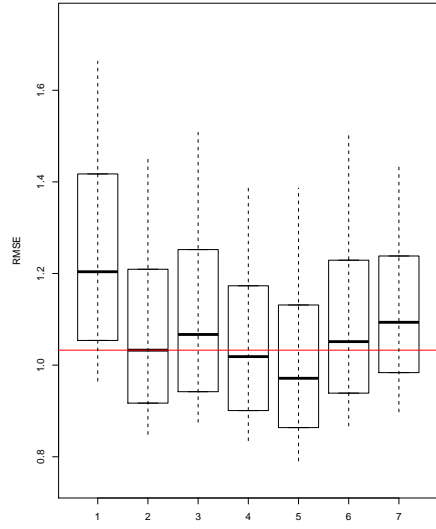
    if (indexName == "Cor. deseasonal") {
      pred2 <- scaleGrid(pred,time.frame = "daily",window.width = 31) %>% redim(drop=TRUE)
      index[i,,] <- valueMeasure(yt2,pred2,measure.code="ts.rp")$Measure$Data
      ylim <- c(0.9,1)
    }
  }
}
```



```

if (indexName == "Stand dev.") {
  index[i,,] <- valueMeasure(yt,pred,measure.code="ratio",index.code="Var")$Measure$Data
  index[i,,] <- index[i,,] ** (1/2)
  ylim <- c(0.96,1.03)
}
if (indexName == "bias") {
  index[i,,] <- valueMeasure(yt,pred,measure.code="bias",index.code="Mean")$Measure$Data
  ylim <- c(-0.2,0.5)
}
if (indexName == "bias02") {
  index[i,,] <- valueMeasure(yt,pred,measure.code="bias",index.code="P02")$Measure$Data
  ylim <- c(-0.5,1.5)
}
if (indexName == "bias98") {
  index[i,,] <- valueMeasure(yt,pred,measure.code="bias",index.code="P98")$Measure$Data
  ylim <- c(-1,1)
}
}
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)
lines(c(0,8),c(mglm4,mglm4), col = "red")
for (i in 1:datasetspred) lines(c(i,i),perc[,i], lty = 2)
}

```



```
# 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`.

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

  yt2 <- scaleGrid(yt, time.frame = "daily", window.width = 31) %>% redim(drop=TRUE)
  load(paste0("./Data/temperature/predictions_test_", model, ".rda"))
  indexNames <- c("RMSE", "Cor. deseasonal",
                  "bias", "Stand dev.", "bias02", "bias98")
  pplot <- list(); n <- 0
  for (indexName in indexNames) {
    if (indexName == "RMSE") {
      index <- valueMeasure(yt, pred, measure.code="ts.RMSE")$Measure %>% redim()
      at <- seq(0, 2, 0.1); colorbar <- cb
    }
    if (indexName == "Cor.deseasonal") {
      pred2 <- scaleGrid(pred, time.frame = "daily", window.width = 31) %>% redim(drop=TRUE)
      index <- valueMeasure(yt2, pred2, measure.code="ts.rp")$Measure %>% redim()
      at <- seq(0.85, 1, 0.005); colorbar <- cb
    }
    if (indexName == "bias") {
      index <- valueMeasure(yt, pred, measure.code="bias", index.code="Mean")$Measure %>% redim()
      at <- seq(-2, 2, 0.1); colorbar <- cb2
    }
    if (indexName == "Stand dev.") {
      index <- valueMeasure(yt, pred, measure.code="ratio", index.code="Var")$Measure %>% redim()
      index$Data <- index$Data ** (1/2)
      at <- seq(0.8, 1.2, 0.01); colorbar <- cb2
    }
    if (indexName == "bias02") {
      index <- valueMeasure(yt, pred, measure.code="bias", index.code="P02")$Measure %>% redim()
      at <- seq(-2, 2, 0.1); colorbar <- cb2
    }

    if (indexName == "bias98") {
      index <- valueMeasure(yt, pred, measure.code="bias", index.code="P98")$Measure %>% redim()
      at <- seq(-2, 2, 0.1); 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)

```

