#-----------------------------

require(envimaR)

# MANDANTORY: defining the root folder DO NOT change this line

rootDIR = "C:/Users/jomue/edu/geoAI"

source(file.path(envimaR::alternativeEnvi(root\_folder = rootDIR),"src/geo\_AI\_setup.R"),echo = TRUE)

#read data

Koralle<-sf::st\_read("E:/Koralle/images/Coral\_10m\_03\_22.shp")

Koralle

Coral\_ras = raster::stack("E:/Koralle/images/Coral\_10m\_06\_22.tif")

Coral\_ras

names(Coral\_ras)<-c("red","green","blue")

Coral\_ras<-subset(Coral\_ras,c("red","green","blue"))

Koralle<-Koralle[,c(2:6)]

Koralle = sf::st\_transform(Koralle, crs(Coral\_ras))

Coral\_extent <- raster::extent(Coral\_ras)

Koralle <- sf::st\_crop(Koralle, Coral\_extent)

Koralle

# rasterize the coral

rasterized\_vector <- raster::rasterize(Koralle, Coral\_ras[[1]])

rasterized\_vector

rasterized\_vector[is.na(rasterized\_vector[2:4])] <- 1

rasterized\_vector[is.na(rasterized\_vector[1,3,4])] <- 2

rasterized\_vector[is.na(rasterized\_vector[1,2,4])] <- 3

rasterized\_vector[is.na(rasterized\_vector[1:3])] <- 4

rasterized\_vector

raster::writeRaster(rasterized\_vector,

("E:/Koralle/images/Coral\_Mask\_10m\_06\_22\_bleaching.tif"),

overwrite = T)

# divide to training and testing extent

e\_test <- raster::extent(3e+05, 7390240, 320000, 74500000)

e\_train <- raster::extent(320000, 7620000, 409800, 7500040)

coral\_mask\_train <- raster::crop(rasterized\_vector, e\_train)

coral\_dop\_train <- raster::crop(Coral\_ras, e\_train)

coral\_mask\_test <- raster::crop(rasterized\_vector, e\_test)

coral\_dop\_test <- raster::crop(Coral\_ras, e\_test)

raster::writeRaster(

coral\_mask\_test,

("E:/Koralle/images/Coral\_Mask\_bleaching\_10m\_06\_22\_test.tif"),

overwrite = T

)

raster::writeRaster(

coral\_dop\_test,

("E:/Koralle/images/Coral\_Dop\_bleaching\_10m\_06\_22\_test.tif"),

overwrite = T

)

raster::writeRaster(

coral\_mask\_train,

("E:/Koralle/images/Coral\_Mask\_bleaching\_10m\_06\_22\_train.tif"),

overwrite = T

)

raster::writeRaster(

coral\_dop\_train,

("E:/Koralle/images/Coral\_Dop\_bleaching\_10m\_06\_22\_train.tif"),

overwrite = T

)

subset\_ds <-

function(input\_raster,

model\_input\_shape,

path,

targetname = "",

mask = FALSE) {

# determine next number of quadrats in x and y direction, by simple rounding

targetsizeX <- model\_input\_shape[1]

targetsizeY <- model\_input\_shape[2]

inputX <- ncol(input\_raster)

inputY <- nrow(input\_raster)

# determine dimensions of raster so that

# it can be split by whole number of subsets (by shrinking it)

while (inputX %% targetsizeX != 0) {

inputX = inputX - 1

}

while (inputY %% targetsizeY != 0) {

inputY = inputY - 1

}

# determine difference

diffX <- ncol(input\_raster) - inputX

diffY <- nrow(input\_raster) - inputY

# determine new dimensions of raster and crop,

# cutting evenly on all sides if possible

newXmin <- floor(diffX / 2)

newXmax <- ncol(input\_raster) - ceiling(diffX / 2) - 1

newYmin <- floor(diffY / 2)

newYmax <- nrow(input\_raster) - ceiling(diffY / 2) - 1

rst\_cropped <-

suppressMessages(raster::crop(

input\_raster,

raster::extent(input\_raster, newYmin, newYmax, newXmin, newXmax)

))

agg <-

suppressMessages(raster::aggregate(rst\_cropped[[1]], c(targetsizeX, targetsizeY)))

agg[] <- suppressMessages(1:ncell(agg))

agg\_poly <- suppressMessages(raster::rasterToPolygons(agg))

names(agg\_poly) <- "polis"

if (mask) {

lapply(

seq\_along(agg),

FUN = function(i) {

subs <- local({

e1 <- raster::extent(agg\_poly[agg\_poly$polis == i,])

subs <- suppressMessages(raster::crop(rst\_cropped, e1))

})

writePNG(as.array(subs),

target = paste0(path, targetname, i, ".png"))

}

)

}

else{

lapply(

seq\_along(agg),

FUN = function(i) {

subs <- local({

e1 <- raster::extent(agg\_poly[agg\_poly$polis == i,])

subs <- suppressMessages(raster::crop(rst\_cropped, e1))

# rescale to 0-1, for png export

if (mask == FALSE) {

subs <-

suppressMessages((subs - cellStats(subs, "min")) / (cellStats(subs, "max") -

cellStats(subs, "min")))

}

})

writePNG(as.array(subs),

target = paste0(path, targetname, i, ".png"))

}

)

}

rm(subs, agg, agg\_poly)

gc()

return(rst\_cropped)

}

remove\_files <- function(df) {

lapply(

seq(1, nrow(df)),

FUN = function(i) {

local({

fil = df$list\_masks[i]

png = readPNG(fil)

len = length(png)

if (AllEqual(png)) {

file.remove(df$list\_dops[i])

file.remove(df$list\_masks[i])

} else {

}

})

}

)

}

# read training data

coral\_mask\_train <-

raster::stack("E:/Koralle/images/Coral\_Mask\_bleaching\_10m\_06\_22\_train.tif")

coral\_dop\_train <-

raster::stack("E:/Koralle/images/Coral\_Dop\_bleaching\_10m\_06\_22\_train.tif")

# set the size of each image

model\_input\_shape = c(128, 128)

subset\_ds(

input\_raster = coral\_mask\_train,

path = "E:/Koralle/images/Cor\_bleaching\_10m\_06\_22/",

mask = TRUE,

model\_input\_shape = model\_input\_shape

)

subset\_ds(

input\_raster = coral\_dop\_train,

path = "E:/Koralle/images/Dop\_bleaching\_10m\_06\_22/",

mask = FALSE,

model\_input\_shape = model\_input\_shape

)

# list all created files in both folders

list\_dops <-

list.files("E:/Koralle/images/Dop\_bleaching\_10m\_06\_22/",

full.names = TRUE,

pattern = "\*.png")

list\_masks <-

list.files("E:/Koralle/images/Cor\_bleaching\_10m\_06\_22/",

full.names = TRUE,

pattern = "\*.png")

# create a data fram

df = data.frame(list\_dops, list\_masks)

remove\_files(df)

# list the files again

files <- data.frame(

img = list.files(

file.path("E:/Koralle/images/Dop\_bleaching\_10m\_06\_22/"),

full.names = TRUE,

pattern = "\*.png"

),

mask = list.files(

file.path("E:/Koralle/images/Cor\_bleaching\_10m\_06\_22/"),

full.names = TRUE,

pattern = "\*.png"

)

)

# split randomly into training and validation (not testing!!) data sets

set.seed(7)

data <- initial\_split(files, prop = 0.8)

# function to prepare your data set for all further processes

prepare\_ds <-

function(files = NULL,

train,

predict = FALSE,

subsets\_path = NULL,

model\_input\_shape = c(256, 256),

batch\_size = batch\_size,

visual = FALSE) {

if (!predict) {

# function for random change of saturation,brightness and hue,

# will be used as part of the augmentation

spectral\_augmentation <- function(img) {

img <- tf$image$random\_brightness(img, max\_delta = 0.1)

img <-

tf$image$random\_contrast(img, lower = 0.9, upper = 1.1)

img <-

tf$image$random\_saturation(img, lower = 0.9, upper = 1.1)

# make sure we still are between 0 and 1

img <- tf$clip\_by\_value(img, 0, 1)

}

# create a tf\_dataset from the input data.frame

# right now still containing only paths to images

dataset <- tensor\_slices\_dataset(files)

# use dataset\_map to apply function on each record of the dataset

# (each record being a list with two items: img and mask), the

# function is list\_modify, which modifies the list items

# 'img' and 'mask' by using the results of applying decode\_png on the img and the mask

# -> i.e. pngs are loaded and placed where the paths to the files were (for each record in dataset)

dataset <-

dataset\_map(dataset, function(.x)

list\_modify(

.x,

img = tf$image$decode\_png(tf$io$read\_file(.x$img)),

mask = tf$image$decode\_png(tf$io$read\_file(.x$mask))

))

# convert to float32:

# for each record in dataset, both its list items are modified

# by the result of applying convert\_image\_dtype to them

dataset <-

dataset\_map(dataset, function(.x)

list\_modify(

.x,

img = tf$image$convert\_image\_dtype(.x$img, dtype = tf$float32),

mask = tf$image$convert\_image\_dtype(.x$mask, dtype = tf$float32)

))

# data augmentation performed on training set only

if (train) {

# augmentation 1: flip left right, including random change of

# saturation, brightness and contrast

# for each record in dataset, only the img item is modified by the result

# of applying spectral\_augmentation to it

augmentation <-

dataset\_map(dataset, function(.x)

list\_modify(.x, img = spectral\_augmentation(.x$img)))

#...as opposed to this, flipping is applied to img and mask of each record

augmentation <-

dataset\_map(augmentation, function(.x)

list\_modify(

.x,

img = tf$image$flip\_left\_right(.x$img),

mask = tf$image$flip\_left\_right(.x$mask)

))

dataset\_augmented <-

dataset\_concatenate(augmentation, dataset)

# augmentation 2: flip up down,

# including random change of saturation, brightness and contrast

augmentation <-

dataset\_map(dataset, function(.x)

list\_modify(.x, img = spectral\_augmentation(.x$img)))

augmentation <-

dataset\_map(augmentation, function(.x)

list\_modify(

.x,

img = tf$image$flip\_up\_down(.x$img),

mask = tf$image$flip\_up\_down(.x$mask)

))

dataset\_augmented <-

dataset\_concatenate(augmentation, dataset\_augmented)

# augmentation 3: flip left right AND up down,

# including random change of saturation, brightness and contrast

augmentation <-

dataset\_map(dataset, function(.x)

list\_modify(.x, img = spectral\_augmentation(.x$img)))

augmentation <-

dataset\_map(augmentation, function(.x)

list\_modify(

.x,

img = tf$image$flip\_left\_right(.x$img),

mask = tf$image$flip\_left\_right(.x$mask)

))

augmentation <-

dataset\_map(augmentation, function(.x)

list\_modify(

.x,

img = tf$image$flip\_up\_down(.x$img),

mask = tf$image$flip\_up\_down(.x$mask)

))

dataset\_augmented <-

dataset\_concatenate(augmentation, dataset\_augmented)

}

# shuffling on training set only

# unsauber

if (!visual) {

if (train) {

dataset <-

dataset\_shuffle(dataset\_augmented, buffer\_size = batch\_size \* 256)

}

# train in batches; batch size might need to be adapted depending on

# available memory

dataset <- dataset\_batch(dataset, batch\_size)

}

if (visual) {

dataset <- dataset\_augmented

}

# output needs to be unnamed

dataset <- dataset\_map(dataset, unname)

} else{

# make sure subsets are read in in correct order

# so that they can later be reassembled correctly

# needs files to be named accordingly (only number)

o <-

order(as.numeric(tools::file\_path\_sans\_ext(basename(

list.files(subsets\_path)

))))

subset\_list <- list.files(subsets\_path, full.names = T)[o]

dataset <- tensor\_slices\_dataset(subset\_list)

dataset <-

dataset\_map(dataset, function(.x)

tf$image$decode\_png(tf$io$read\_file(.x)))

dataset <-

dataset\_map(dataset, function(.x)

tf$image$convert\_image\_dtype(.x, dtype = tf$float32))

dataset <- dataset\_batch(dataset, batch\_size)

dataset <- dataset\_map(dataset, unname)

}

}

# one more parameter

batch\_size = 8

# prepare data for training

training\_dataset <-

prepare\_ds(

training(data),

train = TRUE,

predict = FALSE,

model\_input\_shape = model\_input\_shape,

batch\_size = batch\_size

)

# also prepare validation data

validation\_dataset <-

prepare\_ds(

testing(data),

train = FALSE,

predict = FALSE,

model\_input\_shape = model\_input\_shape,

batch\_size = batch\_size

)

# we first get a all our training data

it <- as\_iterator(training\_dataset)

it <- iterate(it)

# head(it)

# we convert our data to an array and also subset our iterator e.g.

# with the 4th batch ([[4]]) of the images ([[1]])

im <-as.array(it[[4]][[1]])

# then we subset just take the first image out of our batch

im <- im[1,,,]

# and plot it

plot(as.raster(im))



# and for the according mask it is almost the same

ma <-as.array(it[[4]][[2]])

ma <- ma[1,,,]

plot(as.raster(ma))



#U-Net

# function to build a U-Net

# of course it is possible to change the input\_shape

get\_unet\_128 <- function(input\_shape = c(128, 128, 3),

num\_classes = 1) {

inputs <- layer\_input(shape = input\_shape)

# 128

down1 <- inputs %>%

layer\_conv\_2d(filters = 64,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 64,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu")

down1\_pool <- down1 %>%

layer\_max\_pooling\_2d(pool\_size = c(2, 2), strides = c(2, 2))

# 64

down2 <- down1\_pool %>%

layer\_conv\_2d(filters = 128,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 128,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu")

down2\_pool <- down2 %>%

layer\_max\_pooling\_2d(pool\_size = c(2, 2), strides = c(2, 2))

# 32

down3 <- down2\_pool %>%

layer\_conv\_2d(filters = 256,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 256,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu")

down3\_pool <- down3 %>%

layer\_max\_pooling\_2d(pool\_size = c(2, 2), strides = c(2, 2))

# 16

down4 <- down3\_pool %>%

layer\_conv\_2d(filters = 512,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 512,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu")

down4\_pool <- down4 %>%

layer\_max\_pooling\_2d(pool\_size = c(2, 2), strides = c(2, 2))

# # 8

center <- down4\_pool %>%

layer\_conv\_2d(filters = 1024,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 1024,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu")

# center

up4 <- center %>%

layer\_upsampling\_2d(size = c(2, 2)) %>%

{

layer\_concatenate(inputs = list(down4, .), axis = 3)

} %>%

layer\_conv\_2d(filters = 512,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 512,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 512,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu")

# 16

up3 <- up4 %>%

layer\_upsampling\_2d(size = c(2, 2)) %>%

{

layer\_concatenate(inputs = list(down3, .), axis = 3)

} %>%

layer\_conv\_2d(filters = 256,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 256,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 256,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu")

# 32

up2 <- up3 %>%

layer\_upsampling\_2d(size = c(2, 2)) %>%

{

layer\_concatenate(inputs = list(down2, .), axis = 3)

} %>%

layer\_conv\_2d(filters = 128,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 128,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 128,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu")

# # 64

up1 <- up2 %>%

layer\_upsampling\_2d(size = c(2, 2)) %>%

{

layer\_concatenate(inputs = list(down1, .), axis = 3)

} %>%

layer\_conv\_2d(filters = 64,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 64,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu") %>%

layer\_conv\_2d(filters = 64,

kernel\_size = c(3, 3),

padding = "same") %>%

layer\_activation("relu")

# 128

classify <- layer\_conv\_2d(

up1,

filters = num\_classes,

kernel\_size = c(1, 1),

activation = "sigmoid"

)

model <- keras\_model(inputs = inputs,

outputs = classify)

return(model)

}

unet\_model <- get\_unet\_128()

# compile the model

unet\_model %>% compile(

optimizer = optimizer\_adam(learning\_rate = 0.0001),

loss = "binary\_crossentropy",

metrics = "accuracy"

)

# train the model

hist <- unet\_model %>% fit(

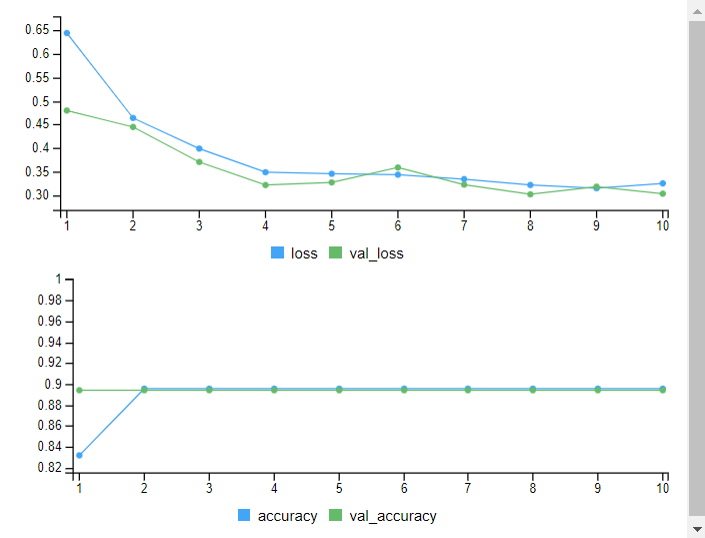
training\_dataset,

validation\_data = validation\_dataset,

epochs = 10,

verbose = 1

)

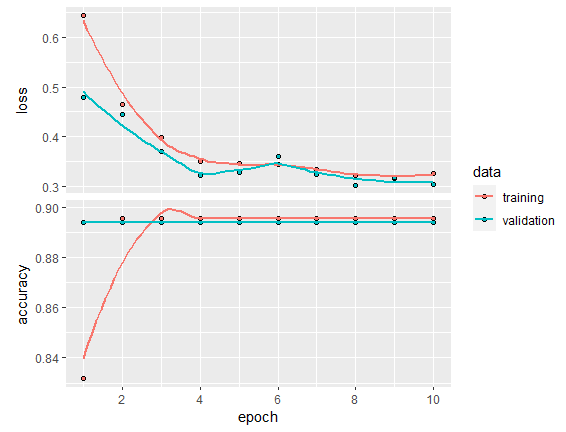


# save the model

unet\_model %>% save\_model\_hdf5(file.path("E:/Koralle/images/models/", "unet\_corals\_10m\_06\_22\_bleaching.hdf5"), overwrite=T)

unet\_model

plot(hist)



# load the test data

coral\_mask\_test <-

stack("E:/Koralle/images/Coral\_Mask\_bleaching\_10m\_06\_22\_test.tif")

coral\_dop\_test <-

stack("E:/Koralle/images/Coral\_Dop\_bleaching\_10m\_06\_22\_test.tif")

target\_rst <-

subset\_ds(

input\_raster = coral\_mask\_test,

path = "E:/Koralle/images/Cor\_test\_bleaching\_10m\_06\_22/",

mask = TRUE,

model\_input\_shape = model\_input\_shape

)

subset\_ds(

input\_raster = coral\_dop\_test,

path = "E:/Koralle/images/Dop\_test\_bleaching\_10m\_06\_22/",

mask = FALSE,

model\_input\_shape = model\_input\_shape

)

# write the target\_rst to later rebuild your image

writeRaster(

target\_rst,

file.path("E:/Koralle/images/models/model\_test\_10m\_06\_22\_bleaching/","coral\_mask\_bleaching\_10m\_06\_22\_test\_target.tif"),

overwrite = T

)

# list and prepare the files again

test\_file <- data.frame(

img = list.files(

file.path("E:/Koralle/images/Dop\_test\_bleaching\_10m\_06\_22"),

full.names = T,

pattern = "\*.png"

),

mask = list.files(

file.path("E:/Koralle/images/Cor\_test\_bleaching\_10m\_06\_22"),

full.names = T,

pattern = "\*.png"

)

)

testing\_dataset <-

prepare\_ds(

test\_file,

train =FALSE,

predict = FALSE,

model\_input\_shape = model\_input\_shape,

batch\_size = batch\_size

)

# load a U-Net

unet\_model <-

load\_model\_hdf5(file.path("E:/Koralle/images/models/", "unet\_corals\_10m\_06\_22\_bleaching.hdf5"),

compile = TRUE)

# evaluate the model with test set

ev <- unet\_model$evaluate(testing\_dataset)

# prepare data for prediction

prediction\_dataset <-

prepare\_ds(

predict = TRUE,

subsets\_path = paste0(file.path("E:/Koralle/images/Dop\_test\_bleaching\_10m\_06\_22/")),

model\_input\_shape = model\_input\_shape,

batch\_size = batch\_size

)

# get sample of data from testing data

t\_sample <-

floor(runif(n = 5, min = 1, max = nrow(test\_file)))

# simple visual comparison of mask, image and prediction

for (i in t\_sample) {

png\_path <- test\_file

png\_path <- png\_path[i,]

img <- image\_read(png\_path[, 1])

mask <- image\_read(png\_path[, 2])

pred <-

image\_read(as.raster(predict(object = unet\_model, testing\_dataset)[i, , ,]))

out <- image\_append(c(

image\_annotate(

mask,

"Mask",

size = 10,

color = "black",

boxcolor = "white"

),

image\_annotate(

img,

"Original Image",

size = 10,

color = "black",

boxcolor = "white"

),

image\_annotate(

pred,

"Prediction",

size = 10,

color = "black",

boxcolor = "white"

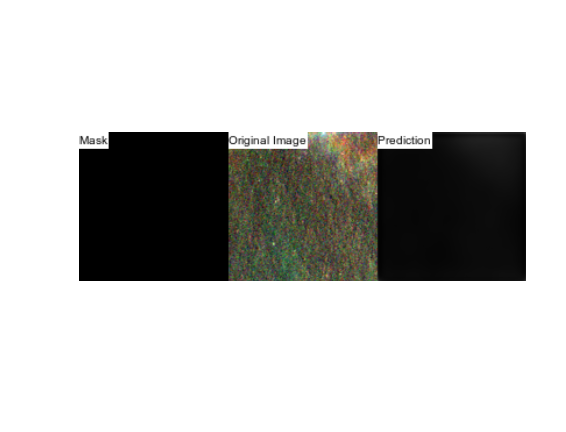
)

))

plot(out)

}





# function to rebuild your image

rebuild\_img <-

function(pred\_subsets,

out\_path,

target\_rst,

model\_name) {

subset\_pixels\_x <- ncol(pred\_subsets[1, , , ])

subset\_pixels\_y <- nrow(pred\_subsets[1, , , ])

tiles\_rows <- nrow(target\_rst) / subset\_pixels\_y

tiles\_cols <- ncol(target\_rst) / subset\_pixels\_x

# load target image to determine dimensions

target\_stars <- st\_as\_stars(target\_rst, proxy = F)

#prepare subfolder for output

result\_folder <- paste0(out\_path, model\_name)

if (dir.exists(result\_folder)) {

unlink(result\_folder, recursive = T)

}

dir.create(path = result\_folder)

# for each tile, create a stars from corresponding predictions,

# assign dimensions using original/target image, and save as tif:

for (crow in 1:tiles\_rows) {

for (ccol in 1:tiles\_cols) {

i <- (crow - 1) \* tiles\_cols + (ccol - 1) + 1

dimx <-

c(((ccol - 1) \* subset\_pixels\_x + 1), (ccol \* subset\_pixels\_x))

dimy <-

c(((crow - 1) \* subset\_pixels\_y + 1), (crow \* subset\_pixels\_y))

cstars <- st\_as\_stars(t(pred\_subsets[i, , , 1]))

attr(cstars, "dimensions")[[2]]$delta = -1

#set dimensions using original raster

st\_dimensions(cstars) <-

st\_dimensions(target\_stars[, dimx[1]:dimx[2], dimy[1]:dimy[2]])[1:2]

write\_stars(cstars, dsn = paste0(result\_folder, "/\_out\_", i, ".tif"))

}

}

starstiles <-

as.vector(list.files(result\_folder, full.names = T), mode = "character")

sf::gdal\_utils(

util = "buildvrt",

source = starstiles,

destination = paste0(result\_folder, "/mosaic.vrt")

)

sf::gdal\_utils(

util = "warp",

source = paste0(result\_folder, "/mosaic.vrt"),

destination = paste0(result\_folder, "/mosaic.tif")

)

}

#load target raster

target\_rst <- raster(file.path("E:/Koralle/images/models/model\_test\_10m\_06\_22\_bleaching/","coral\_mask\_bleaching\_10m\_06\_22\_test\_target.tif"), overwrite=T)

# make the actual prediction

pred\_subsets <- predict(object = unet\_model, x = prediction\_dataset)

# name your output path

model\_name <- "unet\_abc\_bleaching\_10m\_06\_22"

# rebuild .tif from each patch

rebuild\_img(

pred\_subsets = pred\_subsets,

out\_path = paste0(file.path("E:/Koralle/images/prediction/", "/")),

target\_rst = target\_rst,

model\_name = model\_name

)