```
#-----------------------------
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_bleaching_10m_2022.shp")
Koralle
Coral_ras = raster::stack("E:/Koralle/images/Coral_2018.tif")
Coral_ras
names(Coral_ras)<-c("red","green","blue")
Coral_ras<-subset(Coral_ras,c("red","green","blue"))
Koralle = sf::st_transform(Koralle, crs(Coral_ras))
Koralle<-Koralle[,c(2:6)]

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]])

# reclassify to 0 and 1
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
#save
raster::writeRaster(rasterized_vector,
                    ("E:/Koralle/images/Coral_Mask_2018_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_2018_test.tif"),
  overwrite = T
)

raster::writeRaster(
  coral_dop_test,
  ("E:/Koralle/images/Coral_Dop_bleaching_2018_test.tif"),
  overwrite = T
)

raster::writeRaster(
  coral_mask_train,
  ("E:/Koralle/images/Coral_Mask_bleaching_2018_train.tif"),
  overwrite = T
)

raster::writeRaster(
  coral_dop_train,
  ("E:/Koralle/images/Coral_Dop_bleaching_2018_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
```

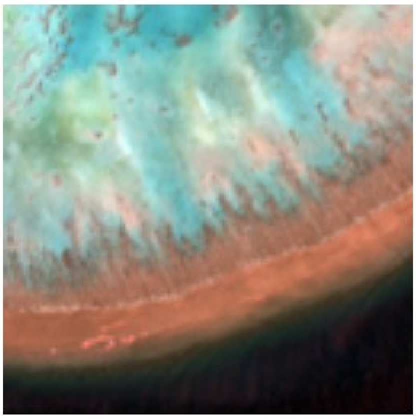
```
            if (mask == FALSE) {
              subs <-
                suppressMessages((subs - cellStats(subs, "min")) /
(cellStats(subs, "max") -
                                                  cellStats(subs, "min")))
            }
          })
          writePNG(as.array(subs),
                   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_2018_train.tif")
coral_dop_train <-
  raster::stack("E:/Koralle/images/Coral_Dop_bleaching_2018_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_2018/",
  mask = TRUE,
  model_input_shape = model_input_shape
)


subset_ds(
  input_raster = coral_dop_train,
  path = "E:/Koralle/images/Dop_bleaching_2018/",
  mask = FALSE,
  model_input_shape = model_input_shape
)

# list all created files in both folders
list_dops <-
  list.files("E:/Koralle/images/Dop_bleaching_2018/",
             full.names = TRUE,
             pattern = "*.png")
list_masks <-
  list.files("E:/Koralle/images/Cor_bleaching_2018/",
```

```
              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_2018/"),
    full.names = TRUE,
    pattern = "*.png"
  ),
  mask = list.files(
    file.path("E:/Koralle/images/Cor_bleaching_2018/"),
    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))
```

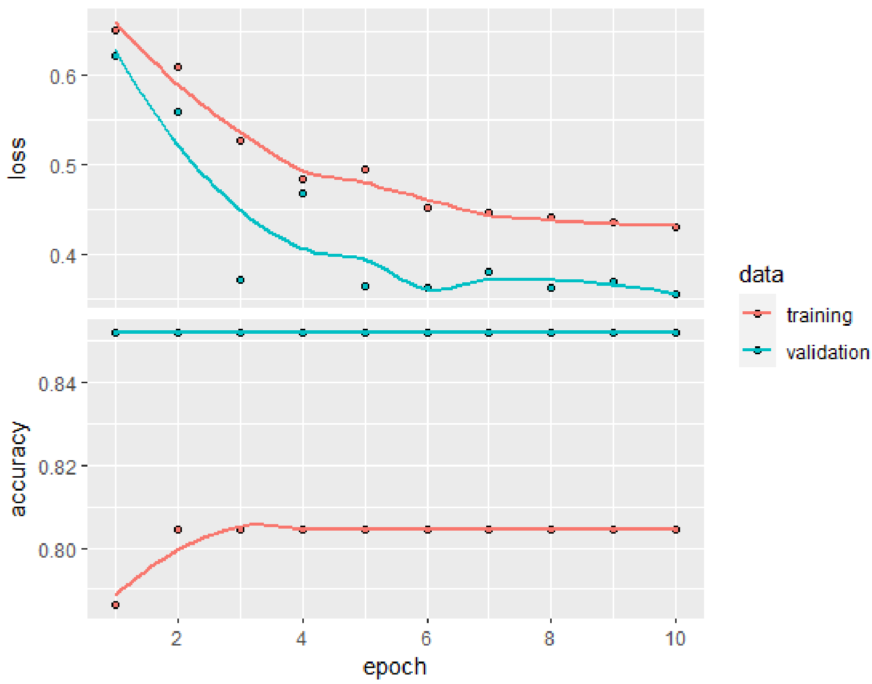


```
# and for the according mask it is almost the same
ma <-as.array(it[[4]][[2]])
ma <- ma[1,,,]
plot(as.raster(ma))
```

```
#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")
```

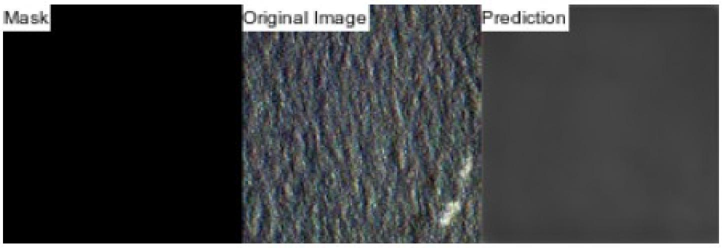
```
# center
up4 <- center %>%
  layer_upsampling_2d(size = c(2, 2)) %>%
  {
    layer_concatenate(inputs = list(down4, .), axis = 3)
  } %>%
  layer_conv_2d(filters = 512,
                kernel_size = c(3, 3),
                padding = "same") %>%
  layer_activation("relu") %>%
  layer_conv_2d(filters = 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") %>%
```

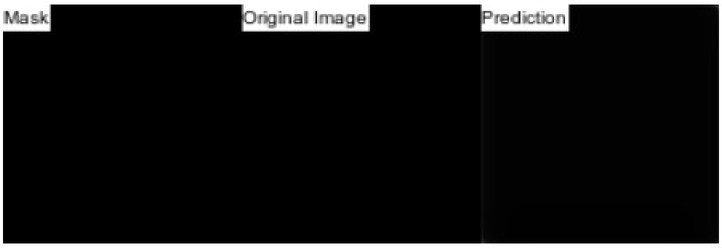
```
    layer_conv_2d(filters = 64,
                  kernel_size = c(3, 3),
                  padding = "same") %>%
    layer_activation("relu") %>%
    layer_conv_2d(filters = 64,
                  kernel_size = c(3, 3),
                  padding = "same") %>%
    layer_activation("relu")
  # 128
  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_2018_bleaching.hdf5"),overwrite=T)

plot(hist)
```

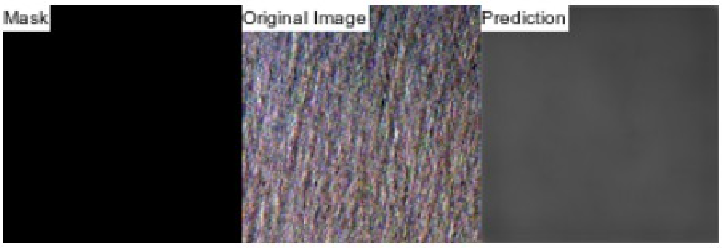


```
# load the test data
coral_mask_test <-
  stack("E:/Koralle/images/Coral_Mask_bleaching_2018_test.tif")
coral_dop_test <-
  stack("E:/Koralle/images/Coral_Dop_bleaching_2018_test.tif")
target_rst <-
  subset_ds(
    input_raster = coral_mask_test,
    path = "E:/Koralle/images/Cor_test_bleaching_2018/",
    mask = TRUE,
    model_input_shape = model_input_shape
  )
subset_ds(
  input_raster = coral_dop_test,
  path = "E:/Koralle/images/Dop_test_bleaching_2018/",
  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_2018_bleaching/","coral_mask_blea
ching_2018_test_target.tif"),
  overwrite = T
)
test_file <- data.frame(
  img = list.files(
    file.path("E:/Koralle/images/Dop_test_bleaching_2018"),
    full.names = T,
    pattern = "*.png"
  ),
  mask = list.files(
    file.path("E:/Koralle/images/Cor_test_bleaching_2018"),
    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_2018_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_2018/")),
    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")
    )
  }
target_rst <-
raster(file.path("E:/Koralle/images/models/model_test_2018_bleaching/","coral_ma
sk_bleaching_2018_test_target.tif"))
# make the actual prediction
pred_subsets <- predict(object = unet_model, x = prediction_dataset)
# name your output path
model_name <- "unet_abc_bleaching_2018"
# 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
)
```