Package ‘geodl’

September 24, 2023

Type Package

Title Geospatial semantic segmentation with torch and terra

Version 0.1.0

Date 2023-08-03

Author c(person(family=``Maxwell'', given=``Aaron'',
email=``Aaron.Maxwell@mail.wvu.edu'', role=c(``aut'', ``cre''))

Maintainer Aaron E. Maxwell <Aaron.Maxwell@mail.wvu.edu>

Description This package provides utilities and functions for semantic segmentation of geospatial data using convolutional neural network-based deep learning. Functions allow for creating masks, image chips, data frames listing image chips in a directory, and datasets for use within data loaders. A basic UNet architecture is provided, and more UNet-like architectures will be made available in later releases. Dice and Dice-like loss metrics are also avaialble along with F1-score, recall, and precision assessment metrics implemented with luz. Trained models can be used to predict to spatial data without the need to generate chips from larger spatial extents. The package relies on torch for implementing deep learning, which does not require the installation or a Pyton environment. Raster geospatial data are handled with terra. Models can be trained using a CUDA-enabled GPU; however, multi-GPU training is not supported by terra. Both binary and multiclass models can be trained.

Depends R (>= 4.1)

Imports dplyr,
terra,
diffeR,
caret,
rfUtilities,
MultiscaleDTM

License GPL (>= 3)

NeedsCompilation no

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

R topics documented:

`assessPnts` *assessPnts*

Description

Assess semantic segmentation model using point locations

Usage

```
assessPnts(
  reference,
  predicted,
  multiclass = TRUE,
  mappings = levels(as.factor(reference)),
  positive_case = mappings[1]
)
```

Arguments

`reference` Data frame column or vector of reference classes.

`predicted` Data frame column or vector of predicted classes.

`multiclass` TRUE or FALSE. If more than two classes are differentiated, use TRUE. If only two classes are differentiated and there are positive and background/negative classes, use FALSE. Default is TRUE.

`mappings` Vector of factor level names. These must be in the same order as the factor levels so that they are correctly matched to the correct category. If no mappings are provided, then the factor levels are used by default.

`positive_case`	Factor level associated with the positive case for a binary classification. Default is the second factor level. This argument is not used for multiclass classification.

Details

This function will generate a set of summary metrics when provided reference and predicted classes. For a multiclass classification problem a confusion matrix is produced with the columns representing the reference data and the rows representing the predictions. The following metrics are calculated: overall accuracy (OA), 95% confidence interval for OA (OAU and OAL), the Kappa statistic, map image classification efficacy (MICE), average class user's accuracy (aUA), average class producer's accuracy (aPA), average class F1-score, overall error (Error), allocation disagreement (Allocation), quantity disagreement (Quantity), exchange disagreement (Exchange), and shift disagreement (shift). For average class user's accuracy, producer's accuracy, and F1-score, macro-averaging is used where all classes are equally weighted. For a multiclass classification all class user's and producer's accuracies are also returned.

For a binary classification problem, a confusion matrix is returned along with the following metrics: overall accuracy (OA), overall accuracy 95% confidence interval (OAU and OAL), the Kappa statistic (Kappa), map image classification efficacy (MICE), precision (Precision), recall (Recall), F1-score (F1), negative predictive value (NPV), specificity (Specificity), overall error (Error), allocation disagreement (Allocation), quantity disagreement (Quantity), exchange disagreement (Exchange), and shift disagreement (shift).

Results are returned as a list object. This function makes use of the caret, diffeR, and rfUtilities packages.

Value

List object containing the resulting metrics. For multiclass assessment, the confusion matrix is provided in the $ConfusionMatrix object, the aggregated metrics are provided in the $Metrics object, class user's accuracies are provided in the $UsersAccs object, class producer's accuracies are provided in the $ProducersAccs object, and the list of classes are provided in the $Classes object. For a binary classification, the confusion matrix is provided in the $ConfusionMatrix object, the metrics are provided in the $Metrics object, the classes are provided in the $Classes object, and the positive class label is provided in the $PositiveCase object.

`assessRaster` *assessRaster*

Description

Assess semantic segmentation model using categorical raster grids (wall-to-wall reference data and predictions)

Usage

```
assessRaster(
  reference,
  predicted,
  multiclass = TRUE,
  mappings,
  positive_case = mappings[2]
)
```

Arguments

`reference`	Single-band, categorical spatRaster object representing the reference labels. Note that the reference and predicted data must have the same extent, number of rows and columns, and coordinate reference system.
`predicted`	Single-band, categorical spatRaster object representing the predicted labels. Note that the reference and predicted data must have the ' same extent, number of rows and columns, and coordinate reference system.
`mappings`	Vector of factor level names. These must be in the same order as the factor levels so that they are correctly matched to the correct category. If no mappings are provided, then the factor levels are used by default. This parameter can be especially useful when using raster data as input as it allows the grid codes to be associated with more meaningful labels.
`positive_case`	Factor level associated with the positive case for a binary classification. Default is the second factor level. This argument is not used for multiclass classification.

Details

This function will generate a set of summary metrics when provided reference and predicted classes. For a multiclass classification problem a confusion matrix is produced with the columns representing the reference data and the rows representing the predictions. The following metrics are calculated: overall accuracy (OA), 95% confidence interval for OA (OAU and OAL), the Kappa statistic, map image classification efficacy (MICE), average class user's accuracy (aUA), average class producer's accuracy (aPA), average class F1-score, overall error (Error), allocation disagreement (Allocation), quantity disagreement (Quantity), exchange disagreement (Exchange), and shift disagreement (shift). For average class user's accuracy, producer's accuracy, and F1-score, macro-averaging is used where all classes are equally weighted. For a multiclass classification all class user's and producer's accuracies are also returned.

For a binary classification problem, a confusion matrix is returned along with the following metrics: overall accuracy (OA), overall accuracy 95% confidence interval (OAU and OAL), the Kappa statistic (Kappa), map image classification efficacy (MICE), precision (Precision), recall (Recall), F1-score (F1), negative predictive value (NPV), specificity (Specificity), overall error (Error), allocation disagreement (Allocation), quantity disagreement (Quantity), exchange disagreement (Exchange), and shift disagreement (shift).

Results are returned as a list object. This function makes use of the caret, diffeR, and rfUtilities packages.

Value

List object containing the resulting metrics. For multiclass assessment, the confusion matrix is provided in the $ConfusionMatrix object, the aggregated metrics are provided in the $Metrics object, class user's accuracies are provided in the $UsersAccs object, class producer's accuracies are provided in the $ProducersAccs object, and the list of classes are provided in the $Classes object. For a binary classification, the confusion matrix is provided in the $ConfusionMatrix object, the overall metrics are provided in the $Metrics object, the classes are provided in the $Classes object, and the positive class label is provided in the $PositiveCase object.

baseUNet *baseUnet*

Description

Define a basic UNet architecture for semantic segmentation.

Usage

```
baseUNet(
  nChn = 3,
  nCls,
  encoderChn = c(16, 32, 64, 128),
  decoderChn = c(128, 64, 32, 16),
  botChn = 256,
  useLeaky = FALSE,
  negative_slope = 0.01
)
```

Arguments

`nChn`	Number of channels, bands, or predictor variables in the input image or raster data. Default is 3.
`nCls`	Number of classes being differentiated. For a binary classification, this can be either 1 or 2. If 2, the problem is treated as a multiclass problem, and a multiclass loss metric should be used.
`encoderChn`	Vector of 4 integers defining the number of output feature maps for each of the four encoder blocks. Default is 16, 32, 64, and 128.
`decoderChn`	Vector of 4 integers defining the number of output feature maps for each of the 4 decoder blocks. Default is 128, 64, 32, and 16.
`botChn`	Number of output feature maps from the bottleneck block. Default is 256.
`useLeaky`	TRUE or FALSE. If TRUE, leaky ReLU activation is used as opposed to ReLU. If FALSE, ReLU is used. Default is FALSE.
`negative_slope`	If useLeaky is TRUE, specifies the negative slope term to use. Default is 0.01.

Details

Define a basic Unet architecture with 4 blocks in the encoder, a bottleneck block, and 4 blocks in the decoder. UNet can accept a variable number of input channels, and the user can define the number of feature maps produced in each encoder and decoder block and the bottleneck. When the UNet is used to predict to new data, it will return either the positive class logit, in the case of a binary classification, or a logit for each class in the case of a multiclass classification.

Value

Instantiated UNet model as subclass of torch::nn_module(). If used to infer new data, will return a tensor of predicted logits.

defineDiceLossFamily *defineDiceLossFamily*

Description

Define a loss metric for binary or multiclass classification based on Dice (class-based).

Usage

```
defineDiceLossFamily(
  nCls,
  smooth = 1,
  mode = "multiclass",
  alpha = 0.5,
  beta = 0.5,
  gamma = 1,
  average = "micro",
  tversky = FALSE,
  focal = FALSE,
  combo = FALSE,
  useWghts = FALSE,
  wghts = c(1, 1),
  ceWght = 1,
  chnDim = TRUE,
  mskLong = TRUE
)
```

Arguments

`smooth`	Smooth factor to aid in stability and to prevent divide-by-zero errors. Default is 1.
`mode`	Either "multiclass" or "binary". Default is "multiclass". If "multiclass", the prediction should be provided as Batch, Channel, Height, Width, where the channel dimension provides the predicted logit for each class, and the target should be Batch, Channel, Height, Width, where the channel dimension provides the index for the correct class. Script assumes class indices start at 0 as opposed to 1. If "binary", the prediction should be provided as Batch, Channel, Height, Width, where the channel dimension provides the predicted logit for the positive class, and the target should be Batch, Channel, Height, Width, where the channel dimension provides the index for the correct class (0 = negative, 1 = Positive). If the target does not included the channel dimension (i.e. Batch, Height, Width), the chnDim argument should be set to FALSE, which will force the script to add the channel dimension. It is best to provide targets in a torch_long data type. If not, the script will convert the targets to long. Predictions should be provided as logits, and a softmax or sigmoid activation should not be applied. The data type for the targets should be torch_float32.
`alpha`	Alpha parameter for false positives in Tversky calculation. This is ignored if Dice is calculated. The default is 0.5.
`beta`	Beta parameter for false negatives in Tversky calculation. This is ignored if Dice is calculated. The default is 0.5

`gamma` Gamma parameter if Focal Tversky or Focal Dice is calculated. Ignored if focal loss is not used. Default is 1.

`average` Either "micro" or "macro". Class averaging method applied for multiclass classification. If "micro", classes are weighted relative to their abundance in the target data. If "macro", classes are equally weighted in the calculation. Default is "micro".

`tversky` TRUE or FALSE. Whether to calculate Tversky as opposed to Dice loss. If TRUE, Tversky is calculated. If FALSE, Dice is calculated. Default is FALSE.

`focal` TRUE or FALSE. Whether to calculate a Focal Dice or Focal Tversky loss. If FALSE, the gamma parameter is ignored. Default is FALSE.

`combo` TRUE or FALSE. Whether to calculate a combo loss using Dice/Tversky + binary entropy/cross entropy. If TRUE, a combo loss is calculated. If FALSE, a combo loss is not calculated. Default is FALSE.

`useWghts` TRUE or FALSE. Default is FALSE. If TRUE, class weights will be applied in the calculation of cross entropy loss and macro-average Dice or Tversky loss. This setting does not impact micro-averaged Dice or Tversky loss. If TRUE, the wght argument must be specified.

`ceWght` If combo is TRUE, defines relative weighting of binary cross entropy/ cross entropy in the loss as (Dice/Tversky) + ceWght*(binary cross entropy/ cross entropy). Ignored if combo is FALSE. Default is 1, or equal weighting in the calculation between the two losses.

`chnDim` TRUE or FALSE. Default is TRUE. If TRUE, assumes the target tensor includes the channel dimension: Batch, Channel, Height, Width. If FALSE, assumes the target tensor does not include the channel dimension: Channel, Height, Width. The script is written such that it expects the channel dimension. So, if FALSE, the script will add the channel dimension as needed.

`mskLong` TRUE or FALSE. Default is TRUE. Data type of target or mask. If the provided target has a data type other than torch_long, this parameter should be set to FALSE. This will cause the script to convert the target to the tensor_long data type as required for the calculations.

`wght` TRUE or FALSE. Default is FALSE. Must be defined if useWhgts is TRUE. A vector of class weights must be provided that has the same length as the number of classes. The vector is converted to a torch tensor within the script.

Details

Allows for defining a Dice-based loss metric including Dice, Focal Dice, Tversky, Focal Tversky, or a combo loss that combines Dice or Tversky loss with binary cross entropy or cross entropy loss. This is implemented as a subclass of torch::nn_module() that uses the geodl::dice_loss_family() function internally. Must be instantiated before use.

Value

Loss metric for us in training process.

defineSegDataSet *defineSegDataSet*

Description

Instantiate a subclass of torch dataset() function for semantic segmentation

Usage

```
defineSegDataSet(
  chpDF,
  folder,
  normalize = FALSE,
  rescaleFactor = 1,
  mskRescale = 1,
  bands = c(1, 2, 3),
  bMns = 1,
  bSDs = 1,
  mskLong = TRUE,
  chnDim = TRUE,
  doAugs = FALSE,
  maxAugs = 0,
  probVFlip = 0,
  probHFlip = 0,
  probBrightness = 0,
  probContrast = 0,
  probGamma = 0,
  probHue = 0,
  probSaturation = 0,
  brightFactor = c(0.8, 1.2),
  contrastFactor = c(0.8, 1.2),
  gammaFactor = c(0.8, 1.2, 1),
  hueFactor = c(-0.2, 0.2),
  saturationFactor = c(0.8, 1.2)
)
```

Arguments

`chpDF`	Data frame of image chip paths created using makeChipsDF().
`folder`	Full path or path relative to the working directory to the folder containing the image chips and associated masks. You must include the final forward slash in the path (e.g., "C:/data/chips/").
`normalize`	TRUE or FALSE. Whether to apply normalization. If FALSE, bMns and bSDs is ignored. Default is FALSE. If TRUE, you must provide bMns and bSDs.
`rescaleFactor`	A rescaling factor to rescale the bands to 0 to 1. For example, this could be set to 255 to rescale 8-bit data. Default is 1 or no rescaling.
`mskRescale`	Can be used to rescale binary masks that are not scaled from 0 to 1. For example, if masks are scaled from 0 and 255, you can divide by 255 to obtain a 0 to 1 scale. Default is 1 or no rescaling.

`bands` Vector of bands to include. The default is to only include the first 3 bands. If you want to use a different subset of bands, you must provide a vector of band indices here to override the default.

`bMns` Vector of band means. Length should be the same as the number of bands. Normalization is applied before any rescaling within the function.

`bSDs` Vector of band standard deviations. Length should be the same as the number of bands. Normalization is applied before any rescaling within the function.

`mskLong` TRUE or FALSE. Default is TRUE. If TRUE, target tensors will be produced with a tensor_long data type. If FALSE, target tensors will be produced with a tensor_float32 data type. We recommend using TRUE or producing targets with a torch_long data type.

`chnDim` TRUE or FALSE. Default is TRUE. If TRUE, will produce target tensors that include the channel dimension: (N, C, H, W). If FALSE, will produce target tensors that do not include the channel dimension: (C, H, W). We recommend including the target dimension.

`doAugs` TRUE or FALSE. Whether or not to apply data augmentations to combat overfitting. If FALSE, all augmentations parameters are ignored. Data augmentations are generally only applied to the training set. Default is FALSE.

`maxAugs` 0 to 7. Maximum number of random augmentations to apply. Default is 0 or no augmentations. Must be changed if augmentations are desired.

`probVFlip` 0 to 1. Probability of applying vertical flips. Default is 0 or no augmentations. Must be changed if augmentations are desired.

`probHFlip` 0 to 1. Probability of applying horizontal flips. Default is 0 or no augmentations. Must be changed if augmentations are desired.

`probBrightness` 0 to 1. Probability of applying brightness augmentation. Default is 0 or no augmentations. Must be changed if augmentations are desired.

`probContrast` 0 to 1. Probability of applying contrast augmentations. Default is 0 or no augmentations. Must be changed if augmentations are desired.

`probGamma` 0 to 1. Probability of applying gamma augmentations. Default is 0 or no augmentations. Must be changed if augmentations are desired.

`probHue` 0 to 1. Probability of applying hue augmentations. Default is 0 or no augmentations. Must be changed if augmentations are desired.

`probSaturation` 0 to 1. Probability of applying saturation augmentations. Default is 0 or no augmentations. Must be changed if augmentations are desired.

`brightFactor` Vector of smallest and largest brightness adjustment factors. Random value will be selected between these extremes. The default is 0.8 to 1.2. Can be any non negative number. 0 gives a black image, 1 gives the original image, and 2 increases the brightness by a factor of 2.

`contrastFactor` Vector of smallest and largest contrast adjustment factors. Random value will be selected between these extremes. The default is 0.8 to 1.2. Can be any non negative number. 0 gives a solid gray image, 1 gives the original image, and 2 increases the contrast by a factor of 2.

`gammaFactor` Vector of smallest and largest gamma values and gain value for a total of 3 values. Random value will be selected between these extremes. The default gamma value range is 0.8 to 1.2 and the default gain is 1. The gain is not randomly altered, only the gamma. Non negative real number. gamma larger than 1 make the shadows darker, while gamma smaller than 1 make dark regions lighter.

`hueFactor` Vector of smallest and largest hue adjustment factors. Random value will be selected between these extremes. The default is -0.2 to 0.2. Should be in (-0.5, 0.5). 0.5 and -0.5 give complete reversal of hue channel in HSV space in positive and negative direction respectively. 0 means no shift. Therefore, both -0.5 and 0.5 will give an image with complementary colors while 0 gives the original image.

`saturationFactor` Vector of smallest and largest saturation adjustment factors. Random value will be selected between these extremes. The default is 0.8 to 1.2. 0 will give a black and white image, 1‘will give the original image while 2 will enhance the saturation by a factor of 2.

Details

This function instantiates a subclass of the torch dataset() function that loads data generated using the makeChips() or makeChipsMultiClass() functions. Can also define random augmentations to combat overfitting. Note that horizontal and vertical flips will effect the alignment of the image and associated mask chips. As a result, the same augmentation will be applied to both the image in the mask. Changes in brightness, contrast, gamma, hue, and saturation will not be applied to the masks since alignment is not impacted by these transformations.

Value

A dataset object that can be provided to torch::dataloader().

`describeBatch` *describeBatch*

Description

Generate summary information about a batch of image chips and masks.

Usage

```
describeBatch(dataLoader)
```

Arguments

`dataLoader` Instantiated instance of a DataLoader created using torch::dataloader().

Details

The goal of this function is to provide a check of a batch of image chips and associated masks generated by a DataLoader instance using defineSegDataSet(). Summary information includes the batch size (batchSize); image chip data type (imageDataType); mask data type (maskDataType); the shape of each image chip as number of channels, height pixel count, and width pixel count (imageShape); the mask shape (maskShape); image band statistics (imageStats) including mean (mean), median (median), minimum (min), maximum (max), and standard deviation (sd); mask class index statistics (mskStats) including minimum (min) and maximum (max) class indices; and count of pixels in each class in the batch (mskCnts).

Value

List object summarizing a batch of image chips and masks.

`describeChips` *describeChips*

Description

Generate data frame of band summary statistics and class pixel counts

Usage

```
describeChips(
  folder,
  extension = ".tif",
  mode = "All",
  subSample = TRUE,
  numChips = 200,
  numChipsBack = 200,
  subSamplePix = TRUE,
  sampsPerChip = 100
)
```

Arguments

`folder`	Full folder path or folder path relative to the current working directory that holds the image chips and associated masks. You must include the final forward slash in the folder path (e.g., "C:/data/chips/").
`extension`	raster file extension (e.g., ".tif", ".png", ".jpeg", or ".img"). The utilities in this package generate files in ".tif" format, so this is the default. This option is provided if chips are generated using another method.
`mode`	Either "All", "Positive", or "Divided". This should match the settings used in the makeChips() function or be set to "All" if makeChipsMultiClass() is used. Default is "All".
`subSample`	TRUE or FALSE. Whether or not to subsample the image chips to calculate the summary metrics. We recommend using a subset if a large set of chips are being summarized to reduce computational load. The default is TRUE.
`numChips`	If subSample is set to TRUE, this parameter defines the number of chips to subset. The default is 200. This parameter will be ignored if subSample is set to FALSE.
`numChipsBack`	If subSample is set to TRUE and the mode is "Divided", this parameter indicates the number of chips to sample from the background-only samples. The default is 200. This parameter will be ignored if subSample is set to FALSE and mode is not "Divided".
`subSamplePix`	TRUE or FALSE. Whether or not to calculate statistics using a subsample of pixels from each image chip as opposed to all pixels. If a large number of chips are available and/or each chip is large, we suggest setting this argument to TRUE to reduce the computational load. The default is TRUE.
`sampsPerChip`	If subSamplePix is TRUE, this parameters specifies the number of random pixels to sample per chip. The default is 100. If subSamplePix is set to FALSE, this parameter is ignored.

Details

This function will generate a set of summary metrics from image chips and associated masks stored in a directory. For each band, the minimum, 1st quartile, median, mean, 3rd quartile, and maximum values are returned. For mask data, the count of pixels in each class are returned. These summarizations can be useful for data normalization and determining class weightings in loss calculations.

Value

List object containing the summary metrics for each band in the $ImageStats object and the count of pixels by class in the $maskStats object.

`dice_loss_family` *dice_loss_family*

Description

Define a loss metric for binary or multiclass classification based on Dice (function-based).

Usage

```
dice_loss_family(
  pred,
  target,
  nCls,
  smooth = 1,
  mode = "multiclass",
  alpha = 0.5,
  beta = 0.5,
  gamma = 1,
  average = "micro",
  tversky = FALSE,
  focal = FALSE,
  combo = FALSE,
  useWghts = FALSE,
  wghts,
  ceWght,
  chnDim = TRUE,
  mskLong = TRUE
)
```

Arguments

`smooth` Smooth factor to aid in stability and to prevent divide-by-zero errors. Default is 1.

`mode` Either "multiclass" or "binary". Default is "multiclass". If "multiclass", the prediction should be provided as (Batch, Channel, Height, Width), where the channel dimension provides the predicted logit for each class, and the target should be (Batch, Channel, Height, Width), where the channel dimension provides the index for the correct class. Script assumes class indices start at 0 as opposed to 1. If "binary", the prediction should be provided as (Batch, Channel, Height, Width), where the channel dimension provides the predicted logit for the positive class, and the target should be (Batch, Channel, Height, Width), where the channel dimension provides the index for the correct class (0 = negative, 1 = Positive). If the target does not included the channel dimension (i.e. (Batch, Height, Width)), the chnDim argument should be set to FALSE, which will force the script to add the channel dimension. It is best to provide targets in a torch_long data type. If not, the script will convert the targets to long. Predictions should be provided as logits, and a softmax or sigmoid activation should not be applied. The data type for the targets should be torch_float32.

`alpha` Alpha parameter for false positives in Tversky calculation. This is ignored if Dice is calculated. The default is 0.5.

`beta` Beta parameter for false negatives in Tversky calculation. This is ignored if Dice is calculated. The default is 0.5

`gamma` Gamma parameter if Focal Tversky or Focal Dice is calculated. Ignored if focal loss is not used. Default is 1.

`average` Either "micro" or "macro". Class averaging method applied for multiclass classification. If "micro", classes are weighted relative to their abundance in the target data. If "macro", classes are equally weighted in the calculation. Default is "micro".

`tversky` TRUE or FALSE. Whether to calculate Tversky as opposed to Dice loss. If TRUE, Tversky is calculated. If FALSE, Dice is calculated. Default is FALSE.

`focal` TRUE or FALSE. Whether to calculate a Focal Dice or Focal Tversky loss. If FALSE, the gamma parameter is ignored. Default is FALSE.

`combo` TRUE or FALSE. Whether to calculate a combo loss using Dice/Tversky + binary entropy/cross entropy. If TRUE, a combo loss is calculated. If FALSE, a combo loss is not calculated. Default is FALSE.

`useWghts` TRUE or FALSE. Default is FALSE. If TRUE, class weights will be applied in the calculation of cross entropy loss and macro-average Dice or Tversky loss. This setting does not impact micro-averaged Dice or Tversky loss. If TRUE, the wght argument must be specified.

`ceWght` If combo is TRUE, defines relative weighting of binary cross entropy/ cross entropy in the loss as (Dice/Tversky) + ceWght*(binary cross entropy/ cross entropy). Ignored if combo is FALSE. Default is 1, or equal weighting in the calculation between the two losses.

`chnDim` TRUE or FALSE. Default is TRUE. If TRUE, assumes the target tensor includes the channel dimension: (Batch, Channel, Height, Width). If FALSE, assumes the target tensor does not include the channel dimension: (Channel, Height, Width). The script is written such that it expects the channel dimension. So, if FALSE, the script will add the channel dimension as needed.

`mskLong` TRUE or FALSE. Default is TRUE. Data type of target or mask. If the provided target has a data type other than torch_long, this parameter should be set to FALSE. This will cause the script to convert the target to the tensor_long data type as required for the calculations.

`wght` TRUE or FALSE. Default is FALSE. Must be defined if useWhgts is TRUE. A vector of class weights must be provided that has the same length as the number of classes. The vector is converted to a torch tensor within the script.

Details

Allows for defining a Dice-based loss metric including Dice, Focal Dice, Tversky, Focal Tversky, or a combo loss that combines Dice or Tversky loss with binary cross entropy or cross entropy loss. This is a functional implementation that is called internally in the defineDiceFamilyLoss() class-based implementation.

Value

Loss metric for us in training process.

---

luz_metric_f1score *luz_metric_f1score*

Description

luz_metric function to calculate the F1-score

Usage

```
luz_metric_f1score(mode = "multiclass", average = "micro", smooth = 1)
```

Arguments

mode	Either "binary" or "multiclass". If "binary", only the logit for positive class prediction should be provided. If both the positive and negative or background class probability is provided for a binary classification, use the "multiclass" mode.
average	Either "micro" or "macro". Whether to use micro- or macro-averaging for multiclass metric calculation. Ignored when mode is "binary". Default is "micro"
preds	Tensor of class predicted probabilities with shape Batch, Class Logits, Height, Width for a multiclass classification. For a binary classification, you can provide logits for the positive class as Batch, Positive Class Logit, Height, Width or Batch, Height, Width.
target	Tensor of target class labels with shape Batch, Class Indices, Height, Width for a multiclass classification. For a binary classification, you can provide targets as Batch, Positive Class Index, Height, Width or Batch, Height, Width. For binary classification, the class index must be 1 for the positive class and 0 for the background case.

Details

Calculates F1-score based on luz_metric() for use within training and validation loops.

Value

Calculated metric return as a base-R vector as opposed to tensor

luz_metric_precision *luz_metric_recall*

Description

luz_metric function to calculate precision

Usage

```
luz_metric_precision(mode = "multiclass", average = "micro", smooth = 1)
```

Arguments

mode	Either "binary" or "multiclass". If "binary", only the logit for positive class prediction should be provided. If both the positive and negative or background class probability is provided for a binary classification, use the "multiclass" mode.
average	Either "micro" or "macro". Whether to use micro- or macro-averaging for multiclass metric calculation. Ignored when mode is "binary". Default is "micro"
preds	Tensor of class predicted probabilities with shape Batch, Class Logits, Height, Width for a multiclass classification. For a binary classification, you can provide logits for the positive class as Batch, Positive Class Logit, Height, Width or Batch, Height, Width.
target	Tensor of target class labels with shape Batch, Class Indices, Height, Width for a multiclass classification. For a binary classification, you can provide targets as Batch, Positive Class Index, Height, Width or Batch, Height, Width. For binary classification, the class index must be 1 for the positive class and 0 for the background case.

Details

Calculates precision based on luz_metric() for use within training and validation loops.

Value

Calculated metric return as a base-R vector as opposed to tensor

luz_metric_recall *luz_metric_recall*

Description

luz_metric function to calculate recall

Usage

```
luz_metric_recall(mode = "multiclass", average = "micro", smooth = 1)
```

Arguments

mode

Either "binary" or "multiclass". If "binary", only the logit for positive class prediction should be provided. If both the positive and negative or background class logits are provided for a binary classification, use the "multiclass" mode.

average

Either "micro" or "macro". Whether to use micro- or macro-averaging for multiclass metric calculation. Ignored when mode is "binary". Default is "micro"

preds

Tensor of class predicted probabilities with shape (Batch, Class Logits, Height, Width) for a multiclass classification. For a binary classification, you should provide logits for just the positive class as (Batch, Positive Class Logit, Height, Width) or (Batch, Height, Width).

target

Tensor of target class labels with shape (Batch, Class Indices, Height, Width) for a multiclass classification. For a binary classification, you should provide targets as (Batch, Positive Class Index, Height, Width) or (Batch, Height, Width). For binary classification, the class index must be 1 for the positive class and 0 for the background case.

Details

Calculates recall based on luz_metric() for use within training and validation loops.

Value

Calculated metric return as a base-R vector as opposed to a tensor

makeChips *makeChips*

Description

Generate image chips from images and associated raster masks

Usage

```
makeChips(
  image,
  mask,
  n_channels = 3,
  size = 256,
  stride_x = 256,
  stride_y = 256,
  outDir,
  mode = "All"
)
```

Arguments

image

Path to input image. Function will generate a SpatRaster object internally. The image and mask must have the same extent, number of rows and columns of pixels, cell size, and coordinate reference system.

`mask` Path to single-band mask. Function will generate a SpatRaster object internally. The image and mask must have the same extent, number of rows and columns if pixels, cell size, and coordinate reference system.

`n_channels` Number of channels in the input image. Default is 3.

`size` Size of image chips as number of rows and columns of pixels. Default is 256.

`stride_x` Stride in the x (columns) direction. Default is 256.

`stride_y` Stride in the y (rows) direction. Default is 256.

`outDir` Full or relative path to the current working directory where you want to write the chips to. Subfolders in this directory will be generated by the function. You must include the final forward slash in the file path (e.g., "C:/data/chips/").

`mode` Either "All", "Positive", or "Divided". Please see the explanations provided above. The default is "All".

Details

This function will generate image and mask chips from an input image and associated raster mask. The chips are written into the defined directory. The number of rows and columns of pixels in each chip are equal to the size argument. If a stride_x and/or stride_y is used that is different from the size argument, resulting chips will either overlap or have gaps between them. In order to not have overlap or gaps, the stride_x and stride_y arguments should be the same as the size argument. Both the image chips and associated masks are written to TIFF format (".tif"). Input data are not limited to three band images. This function is specifically for a binary classification where the positive case is indicated with a cell value of 1 and the background or negative case is indicated with a cell value of 0. If an irregular shaped raster grid is provided, only chips and masks that contain no NA or NoDATA cells will be produced.

Three modes are available. If "All" is used, all image chips will be generated even if they do not contain pixels mapped to the positive case. Within the provided directory, image chips will be written to an "images" folder and masks will be written to a "masks" folder. If "Positive" is used, only chips that have at least 1 pixel mapped to the positive class will be produced. Background-only chips will not be generated. Within the provided directory, image chips will be written to an "images" folder and masks will be written to a "masks" folder. Lastly, if the "Divided" method is used, separate "positive" and "background" folders will be created with "images" and "masks" subfolders. Any chip that has at least 1 pixel mapped to the positive class will be written to the "positive" folder while any chip having only background pixels will be written to the "background" folder.

Value

Image and mask files written to disk in TIFF format. Spatial reference information is not maintained. No R object is returned.

`makeChipsDF` *makeChipsDF*

Description

Create data frame and CSV file listing image chips and associated masks

Usage

```
makeChipsDF(
  folder,
  outCSV,
  extension = ".tif",
  mode = "All",
  shuffle = FALSE,
  saveCSV = FALSE
)
```

Arguments

`folder` Full path or path relative to the working directory to the folder containing the image chips and associated masks. You must include the final forward slash in the path (e.g., "C:/data/chips/").

`outCSV` File name and full path or path relative to the working directory for the resulting CSV file with a ".csv" extension.

`extension` The extension of the image and mask raster data (e.g., ".tif", ".png", ".jpeg", or ".img"). The default is ".tif" since this is the file format used by the utilities in this package. This option is provided if chips are generated using another method.

`mode` Either "All", "Positive", or "Divided". This should match the setting used in the makeChips() function. If the makeChipsMultiClass() function was used, this should be set to "All" or left as the default. The default is "All".

`shuffle` TRUE or FALSE. Whether or not to shuffle the rows in the table. Rows can be shuffled to potentially reduced autocorrelation in the data. The default is FALSE.

`saveCSV` TRUE or FALSE. Whether or not to save the CSV file or just return the data frame. If this is set to FALSE then the outCSV parameter is ignored and no CSV file is generated. The default is FALSE.

Details

This function creates a data frame and, optionally, a CSV file that lists all of the image chips and associated masks in a directory. Three columns are produced. The chpN column provides the name of the chip, the chpPth column provides the path to the chip, and the chpMsk column provides the path to the associated mask. All paths are relative to the input folder as opposed to the full file path so that the results can still be used if the data are copied to a new location on disk or to a new computer.

Value

Data frame with three columns (chp, chpPth, and mskPth) and, optionally, a CSV file written to disk.

makeChipsMultiClass *makeChipsMultiClass*

Description

Generate image chips from images and associated raster masks for multiclass classification

Usage

```
makeChipsMultiClass(
  image,
  mask,
  n_channels = 3,
  hasZero = TRUE,
  size = 256,
  stride_x = 256,
  stride_y = 256,
  outDir
)
```

Arguments

`image`	Path to input image. Function will generate a SpatRaster object internally. The image and mask must have the same extent, number of rows and columns of pixels, cell size, and coordinate reference system.
`mask`	Path to single-band mask. Function will generate a SpatRaster object internally. The image and mask must have the same extent, number of rows and columns of pixels, cell size, and coordinate reference system.
`n_channels`	Number of channels in the input image. Default is 3.
`hasZero`	If the class codes begin at 0 as opposed to 1, this should be set to TRUE. If the class codes start at 1 as opposed to 0, this should be set to FALSE. In the case where the class codes start at 1, each code will be reduced by 1 so that codes start at 0. For example, codes 1,2,3,4 would be converted to 0,1,2,3. This is because most deep learning frameworks expect the class codes to be represented as indices from 0 to n-1 where n is the number of classes. Users should be aware of this manipulation and its impact of class code mappings.
`size`	Size of image chips as number of rows and columns of pixels. Default is 256.
`stride_x`	Stride in the x (columns) direction. Default is 256.
`stride_y`	Stride in the y (rows) direction. Default is 256.
`outDir`	Full or relative path to the current working directory where you want to write the chips to. Subfolders in this directory will be generated by the function. You must include the final forward slash in the file path (e.g., "C:/data/chips/").

Details

This function will generate image and mask chips from an input image and associated raster mask. The chips will be written into the defined directory. The number of rows and columns of pixels per chip are equal to the size argument. If a stride_x and/or stride_y is used that is different from the size argument, resulting chips will either overlap or have gaps between them. In order to not have overlap or gaps, the stride_x and stride_y arguments should be the same as the size argument. Both the image chips and associated masks are written to TIFF format (".tif"). Input data are not limited to three band images. This function is specifically for a multiclass classification. For a binary classification, use the makeChips() function. If an irregular shaped raster grid is provided, only chips and masks that contain no NA or NoDATA cells will be produced.

Within the provided directory, image chips will be written to an "images" folder and masks will be written to a "masks" folder.

Value

Image and mask files written to disk in TIFF format. No R object is returned.

makeMasks *makeMasks*

Description

Make raster mask from input vector data

Usage

```
makeMasks(
  image,
  features,
  crop = FALSE,
  extent,
  field,
  background = 0,
  outImage,
  outMask,
  mode = "Both"
)
```

Arguments

`image`	File name and full path or path relative to working directory for image. Image is converted to a spatRaster internally.
`features`	File name and full path or path relative to working directory for vector mask or label data. A field should be provided that differentiates classes using unique numeric codes as explained above. If the input features use a different coordinate reference system then the input image, the features will be reprojected to match the image. Vector data are converted to a SpatVector object internally.
`crop`	TRUE or FALSE. Whether or not to crop the input image data relative to a defined vector extent. The default is FALSE.
`extent`	File name and full path or path relative to working directory for vector extent data. If the extent uses a different coordinate reference system then the input image, the features will be reprojected to match the image. Vector data are converted to a SpatVector object internally.
`field`	The name of the field in the feature vector data that differentiate classes using a unique numeric code with an integer data type. Field name should be provided as a string.

`background` The numeric value to assign to the background class. The default is 0. If the full spatial extent has labels in the input feature data, no background value will be applied. For binary classification problems, the background should be coded to 0 and the positive case should be coded to 1. It is not necessary to include the background class in the vector feature data.

`outImage` Image output name in TIFF format (".tif") with full path or path relative to working directory for image. This output will only be generated if the mode is set to "Both".

`outMask` Mask output name in TIFF format (".tif") with full path or path relative to working directory for image. Output will be a single-band raster grid of class numeric codes.

`mode` Either "Both" or "Mask". If "Both", a copy of the image will be made along with the generated raster mask. If "Mask", only the mask is produced. If you are experiencing issues with alignment between the image and associated mask, setting the mode to "Both" can alleviate this issue. However, this will result in more data being written to disk.

Details

This function creates a raster mask from input vector data. The cell value is indicated by the field parameter. A unique numeric code should be provided for each class. In the case of a binary classification, 0 should indicate background and 1 should indicate positive. For a multiclass problem, values should be sequential from 0 to n-1, where n is the number of classes, or 1 to n. We recommend using 0 to n-1. If no cropping is applied, the generated raster mask should have the same spatial extent, number of rows of pixels, number of columns of pixels, cell size, and coordinate reference system as the input image.

Value

Single-band raster mask written to disk in TIFF format and, optionally, a copy of the image written to disk. Cropping may be applied as specified. No R objects are returned.

---

makeTerrainDerivatives *makeTerrainDerivatives*

Description

Make three band terrain stack from input digital terrain model

Usage

```
makeTerrainDerivatives(dtm, res, filename)
```

Arguments

`dtm` Input SpatRaster object representing bare earth surface elevations.

`res` Resolution of the grid relative to coordinate reference system units (e.g., meters).

`filename` Name and full path or path relative to working directory for output terrain stack. We recommend saving the data to either TIFF (".tif") or Image (".img") format.

Details

This function creates a three-band raster stack from an input digital terrain model (DTM) of bare earth surface elevations. The first band is a topographic position index (TPI) calculated using a moving window with a 50 m circular radius. The second band is the square root of slope calculated in degrees. The third band is a TPI calculated using an annulus moving window with an inner radius of 2 and outer radius of 5 meters. The TPI values are clamped to a range of -10 to 10 then linearly rescaled from 0 and 1. The square root of slope is clamped to a range of 0 to 10 then linearly rescaled from 0 to 1. Values are provided in floating point.

The stack is described in the following publication:

Maxwell, A.E., W.E. Odom, C.M. Shobe, D.H. Doctor, M.S. Bester, and T. Ore, 2023. Exploring the influence of input feature space on CNN-based geomorphic feature extraction from digital terrain data, Earth and Space Science, 10: e2023EA002845. https://doi.org/10.1029/2023EA002845.

Value

Three-band raster mask written to disk in TIFF format and spatRaster object.

---

predictSpatial *predictSpatial*

Description

Apply a trained semantic segmentation model to predict back to geospatial raster data

Usage

```
predictSpatial(
  imgIn,
  model,
  predOut,
  mode = "multiclass",
  probs = FALSE,
  useCUDA = FALSE,
  nCls,
  chpSize,
  stride_x,
  stride_y,
  crop,
  nChn = 3,
  normalize = FALSE,
  bMns,
  bSDs,
  rescaleFactor = 1
)
```

Arguments

imgIn	Input image to classify. Can be a file path (full or relative to current working directory) or a spatRaster object. Should have the same number of bands as the data used to train the model. Bands must also be in the same order.

`model` Trained model to use to infer to new data.

`predOut` Name of output prediction with full path or path relative to the working directory. Must also include the file extension (e.g., ".tif).

`mode` Either "multiclass" or "binary". Default is "multiclass". If model returns a single logit for the positive case, sould be "binary". If two or more class logits are returned, this shoud be "multiclass".

`probs` TRUE OR FALSE. Whether to generate a "hard" classification or return probabilities for each class. If TRUE and for a binary classification, the positive class probability is returned as a single-band raster grid. If TRUE and for a multiclass classification, all class probabilities are returned as a multiband raster. Probabilities will sum to 1, ignoring rounding error. If FALSE, the predicted class index value is returned. This is the class with the largest predicted logit or softmax/sigmoid probability. Default is FALSE.

`useCUDA` TRUE or FALSE. Whether or not to perform the inference on a GPU. If TRUE, the GPU is used. If FALSE, the CPU is used. Must have access to a CUDA-enabled graphics card. Default is FALSE. Note that using a GPU significantly speeds up inference.

`nCls` Number of classes being differentiated. Should be 1 for a binary classification problem and the number of classes for a multiclass classification problem.

`chpSize` Size of image chips that will be fed through the prediction process. We recommend using the size of the image chips used to train the model. However, this is not strictly necessary.

`stride_x` Stride in the x direction. We recommend using a 50% overlap.

`stride_y` Stride in the y direction. We recommend using a 50% overlap.

`crop` Number of rows and columns to crop from each side of the image chip to reduce edge effects. We recommend at least 20.

`nChn` Number of input channels. Default is 3.

`normalize` TRUE or FALSE. Whether to apply normalization. If FALSE, bMns and bSDs is ignored. Default is FALSE. If TRUE, you must provide bMns and bSDs. This should match the setting used in defineSegDataSet().

`bMns` Vector of band means. Length should be the same as the number of bands. Normalization is applied before any rescaling within the function. This should match the setting used in defineSegDataSet().

`bSDs` Vector of band standard deviations. Length should be the same as the number of bands. Normalization is applied before any rescaling within the function. This should match the setting used in defineSegDataSet().

`rescaleFactor` A rescaling factor to rescale the bands to 0 to 1. For example, this could be set to 255 to rescale 8-bit data. Default is 1 or no rescaling. This should match the setting used in defineSegDataSet().

Details

This function generates a pixel-by-pixel prediction using input data and a trained semantic segmentation model. Can return either hard classifications or class probabilities. Result is written to disk and provided as a spatRaster object.

Value

A spatRast object and a raster grid saved to disk of either predicted class indices or predicted class probabilities.

viewBatch *viewBatch*

Description

Generate image grid of batch of image chips and associated masks created by a DataLoader.

Usage

```
viewBatch(
  dataLoader,
  chnDim = TRUE,
  mskLong = TRUE,
  nRows = 3,
  r = 1,
  g = 2,
  b = 3,
  cNames,
  cColors
)
```

Arguments

dataLoader	Instantiated instance of a DataLoader created using torch::dataloader().
chnDim	TRUE or FALSE. Default is TRUE. If TRUE, assumes the target tensor includes the channel dimension: (Batch, Channel, Height, Width). If FALSE, assumes the target tensor does not include the channel dimension: (Channel, Height, Width). The script is written such that it expects the channel dimension. So, if FALSE, the script will add the channel dimension as needed.
mskLong	TRUE or FALSE. Default is TRUE. Data type of target or mask. If the provided target has a data type other than torch_long, this parameter should be set to FALSE. This will cause the script to convert the target to the tensor_long data type as required for the calculations.
nRows	Number of rows in the image grid. Default is 3.
r	Index of channel to assign to red channel. Default is 1. For gray scale or single-band images, assign the same index to all three bands.
g	Index of channel to assign to green channel. Default is 2. For gray scale or single-band images, assign the same index to all three bands.
b	Index of channel to assign to blue channel. Default is 3. For gray scale or single-band images, assign the same index to all three bands.
cNames	Vector of class names. Must be the same length as number of classes.
cColors	Vector of color values to use to display the masks. Colors are applied based on the order of class indices. Length of vector must be the same as the number of classes.

Details

The goal of this function is to provide a visual check of a batch of image chips and associated masks generated from a DataLoader.

Value

Image grids of example chips and masks loaded from a batch produced by the DataLoader.

`viewBatchPreds` *viewBatchPreds*

Description

Generate image grid of batch of image chips, masks, and predictions for all samples in a DataLoader batch.

Usage

```
viewBatchPreds(
  dataLoader,
  model,
  mode = "multiclass",
  chnDim = TRUE,
  mskLong = TRUE,
  nRows = 4,
  r = 1,
  g = 2,
  b = 3,
  cNames,
  cColors,
  useCUDA = FALSE,
  probs = FALSE
)
```

Arguments

`dataLoader`	Instantiated instance of a DataLoader created using torch::dataloader().
`model`	Fitted model used to predict batch.
`mode`	"multiclass" or "binary". If the prediction returns the postive case logit for a binary classification problem, use "binary". If 2 or more class logits are returned, use "multiclass".
`chnDim`	TRUE or FALSE. Default is TRUE. If TRUE, assumes the target tensor includes the channel dimension: (Batch, Channel, Height, Width). If FALSE, assumes the target tensor does not include the channel dimension: (Channel, Height, Width). The script is written such that it expects the channel dimension. So, if FALSE, the script will add the channel dimension as needed.
`mskLong`	TRUE or FALSE. Default is TRUE. Data type of target or mask. If the provided target has a data type other than torch_long, this parameter should be set to FALSE. This will cause the script to convert the target to the tensor_long data type as required for the calculations.
`nRows`	Number of rows in the image grid. Default is 3.
`r`	Index of channel to assign to red channel. Default is 1. For gray scale or single-band images, assign the same index to all three bands.

g	Index of channel to assign to green channel. Default is 2. For gray scale or single-band images, assign the same index to all three bands.
b	Index of channel to assign to blue channel. Default is 3. For gray scale or single-band images, assign the same index to all three bands.
cNames	Vector of class names. Must be the same length as number of classes.
cColors	Vector of color values to use to display the masks. Colors are applied based on the order of class indices. Length of vector must be the same as the number of classes.
useCUDA	TRUE or FALSE. Default is FALSE. If TRUE, GPU will be used to predict the data batch. If FALSE, predictions will be made on the CPU.
probs	TRUE or FALSE. Default is FALSE. If TRUE, class probabilities will be shown as opposed to the hard classification. If FALSE, hard classification will be shown. For a binary problem, the positive class logit is transformed using a sigmoid function. For multiclass, softmax is used to transform the class logits to probabilities that sum to 1.

Details

The goal of this function is to provide a visual check of predictions for a batch of data.

Value

Image grids of example chips and masks loaded from a batch produced by the DataLoader.

viewChips *viewChips*

Description

Plot a grid of image and/or mask chips

Usage

```
viewChips(
  chpDF,
  folder,
  nSamps = 16,
  mode = "both",
  justPositive = FALSE,
  cCnt = 4,
  rCnt = 4,
  r = 1,
  g = 2,
  b = 3,
  rescale = FALSE,
  rescaleVal = 1,
  cNames,
  cColors,
  useSeed = FALSE,
  seed = 42
)
```

Arguments

`chpDF` Data frame of chip paths created with the makeChipsDF() function.

`folder` Full path or path relative to the working directory to the folder containing the image chips and associated masks. You must include the final forward slash in the path (e.g., "C:/data/chips/").

`nSamps` Number of samples to include in the grid. The default is 16.

`mode` Either "image", "mask" or "both". If "image", a grid is produced for the image chips only. If "mask", a grid is produced for just the masks. If "both", masks are produced for both the image chips and masks. Default is "both".

`cCnt` Number of columns in the grid. Row X Column count must sum to the number of samples being displayed (nSamps). Default is 4.

`rCnt` Number of rows in the grid. Row X Column count must sum to the number of samples being displayed (nSamps). Default is 4.

`r` Band number to map to the red channel. Default is 1.

`g` Band number to map to the green channel. Default is 2.

`b` Band number to map to the red channel. Default is 3.

`cNames` Vector of class names. Class names must be provided.

`cColors` Vector of colors (named colors, hex codes, or rgb()). Colors to use to visualize each class matched based on position in the vector. Colors must be provided.

`useSeed` TRUE or FALSE. Whether or not to set a random seed to make result reproducible. If FALSE, seed is ignored. Default is FALSE.

`seed` Random seed value. Default is 42. This is ignored if useSeed is FALSE.

`justPostitive` TRUE or FALSE. If makeChips() was executed using the mode "Divided", you can choose to only show chips that contained some pixels mapped to the positive class. The default is FALSE. This should be left to the default or set to FALSE if chips were generated using a method other than "Divided".

Details

This function generates a plot of image chips and/or image masks. It serves as a means to visualize chips generated with the makeChips() or makeChipsDF() function. It can used as a check to make sure chips were generated as expected.

Value

Plot of image chip grid (if mode = "image"); plot of mask chip grid (if mode ="mask"); plot of image and mask chip grids (if model = "both").

Index