Package ‘geodl’

August 20, 2024

Type Package

Title Geospatial Semantic Segmentation with Torch and Terra

Version 0.2.0

Date 2027-07-22

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

Description Provides tools for semantic segmentation of geospatial data using convolutional neural network-based deep learning. Utility functions allow for creating masks, image chips, data frames listing image chips in a directory, and DataSets for use within DataLoaders. Additional functions are provided to serve as checks during the data preparation and training process. A UNet architecture can be defined with 4 blocks in the encoder, a bottleneck block, and 4 blocks in the decoder. The 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. Users can also choose to (1) replace all rectified linear unit (ReLU) activation functions with leaky ReLU or swish, (2) implement attention gates along the skip connections, (3) implement squeeze and excitation modules within the encoder blocks, (4) add residual connections within all blocks, (5) replace the bottleneck with a modified atrous spatial pyramid pooling (ASPP) module, and/or (6) implement deep supervision using predictions generated at each stage in the decoder. A unified focal loss framework is implemented after Yeung et al. (2022) <doi:10.1016/j.compmedimag.2021.102026>. We have also implemented assessment metrics using the 'luz' package including F1-score, recall, and precision. Trained models can be used to predict to spatial data without the need to generate chips from larger spatial extents. Functions are available for performing accuracy assessment. The package relies on 'torch' for implementing deep learning, which does not require the installation of a 'Python' environment. Raster geospatial data are handled with 'terra'. Models can be trained using a Compute Unified Device Architecture (CUDA)-enabled graphics processing unit (GPU); however, multi-GPU training is not supported by 'torch' in 'R'.

Depends R (>= 4.1)

Imports torch (>= 0.11.0), torchvision (>= 0.5.1), dplyr (>= 1.1.3), terra (>= 1.7.55), luz (>= 0.4.0), MultiscaleDTM (>= 0.8.2), psych (>= 2.3.3), coro (>= 1.0.3), R6 (>= 2.5.1), readr (>= 2.1.3), rlang (>= 1.1.1)

License GPL (>= 3)

URL `https://github.com/maxwell-geospatial/geodl`,

`https://doi.org/10.31223/X53M6T`,

`https://www.wvview.org/geodl/index.html`

BugReports `https://github.com/maxwell-geospatial/geodl/issues`

NeedsCompilation no

Encoding UTF-8

RoxygenNote 7.3.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

Author Aaron Maxwell [aut, cre, cph],
Sarah Farhadpour [aut],
Srinjoy Das [aut],
Yalin Yang [aut]

Repository CRAN

Date/Publication 2024-08-20 15:00:14 UTC

Contents

`assessDL` *assessDL*

Description

Assess semantic segmentation model using all samples in a torch DataLoader.

Usage

```
assessDL(
  dl,
  model,
  multiclass = TRUE,
  batchSize,
  size,
  nCls,
  cCodes,
  cNames,
  usedDS = FALSE,
  useCUDA = FALSE,
  decimals = 4
)
```

Arguments

`dl`	torch DataLoader object.
`model`	trained model object.
`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. For binary cases, the second class is assumed to be the positive case.
`batchSize`	Batch size used in torch DataLoader.
`size`	Size of image chips in spatial dimensions (e.g., 128, 256, 512).
`nCls`	Number of classes being differentiated.
`cCodes`	Class indices as a vector of integer values equal in length to the number of classes.
`cNames`	Class names as a vector of character strings with a length equal to the number of classes and in the correct order. Class codes and names are matched by position in the cCodes and cNames vectors. For binary case, this argument is ignored, and the first class is called "Negative" while the second class is called "Positive".

`usedDS`	TRUE or FALSE. Whether or not deep supervision was used. Default is FALSE, or it is assumed that deep supervision was not used.
`useCUDA`	TRUE or FALSE. Whether or not to use GPU. Default is FALSE, or GPU is not used. We recommend using a CUDA-enabled GPU if one is available since this will speed up computation.
`decimals`	Number of decimal places to return for assessment metrics. Default is 4.

Details

This function generates a set of summary assessment metrics based on all samples within a torch data loader. Results are returned as a list object. For multiclass assessment, the class names ($Classes), count of samples per class in the reference data ($referenceCounts), count of samples per class in the predictions ($predictionCounts), confusion matrix ($confusionMatrix), aggregated assessment metrics ($aggMetrics) (OA = overall accuracy, macroF1 = macro-averaged class aggregated F1-score, macroPA = macro-averaged class aggregated producer's accuracy or recall, and macroUA = macro-averaged class aggregated user's accuracy or precision), class-level user's accuracies or precisions ($userAccuracies), class-level producer's accuracies or recalls ($producerAccuracies), and class-level F1-scores ($F1Scores). For a binary case, the $Classes, $referenceCounts, $predictionCounts, and $confusionMatrix objects are also returned; however, the $aggMets object is replaced with $Mets, which stores the following metrics: overall accuracy, recall, precision, specificity, negative predictive value (NPV), and F1-score. For binary cases, the second class is assumed to be the positive case.

Value

List object containing the resulting metrics and ancillary information.

Examples

```
## Not run:
metricsOut <- assessDL(dl=testDL,
                       model=model,
                       batchSize=15,
                       size=256,
                       nCls=2,
                       mode="binary",
                       cCodes=c(1,2),
                       cNames=c("Not Mine", "Mine"),
                       usedDS=FALSE,
                       useCUDA=TRUE,
                       decimals=4)

## End(Not run)
```

`assessPnts` *assessPnts*

Description

Assess semantic segmentation model using point locations

Usage

```
assessPnts(
  reference,
  predicted,
  multiclass = TRUE,
  mappings = levels(as.factor(reference)),
  decimals = 4
)
```

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 class 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. For a binary classification, it is assumed that the first class is "Background" and the second class is "Positive".
`decimals`	Number of decimal places to return for assessment metrics. Default is 4.

Details

This function generates a set of summary assessment metrics when provided reference and predicted classes. Results are returned as a list object. For multiclass assessment, the class names ($Classes), count of samples per class in the reference data ($referenceCounts), count of samples per class in the predictions ($predictionCounts), confusion matrix ($confusionMatrix), aggregated assessment metrics ($aggMetrics) (OA = overall accuracy, macroF1 = macro-averaged class aggregated F1-score, macroPA = macro-averaged class aggregated producer's accuracy or recall, amd macroUA = macro-averaged class aggregated user's accuracy or precision), class-level user's accuracies or precisions ($userAccuracies), class-level producer's accuracies or recalls ($producerAccuracies), and class-level F1-scores ($F1Scores). For a binary case, the $Classes, $referenceCounts, $predictionCounts, and $confusionMatrix objects are also returned; however, the $aggMets object is replaced with $Mets, which stores the following metrics: overall accuracy, recall, precision, specificity, negative predictive value (NPV), and F1-score. For binary cases, the second class is assumed to be the positive case.

Value

List object containing the resulting metrics and ancillary information.

Examples

```
#Multiclass example

#Generate example data as data frame of class predictions
inDF <- data.frame(ref = sample(c("Class A", "Class B", "Class C"), 1000, replace=TRUE),
pred = sample(c("Class A", "Class B", "Class C"), 1000, replace=TRUE))

#Calculate metrics
metsOut <- assessPnts(reference=inDF$ref,
                     predicted=inDF$pred,
                     multiclass=TRUE,
                     mappings = c("Class A", "Class B", "Class C"),
                     decimals=4)

print(metsOut)

#Binary example

#Generate example data as data frame of class predictions
inDF <- data.frame(ref = sample(c("Background", "Positive"), 1000, replace=TRUE),
                  pred = sample(c("Background", "Positive"), 1000, replace=TRUE))

#Calculate metrics
metsOut <- assessPnts(reference=inDF$ref,
                     predicted=inDF$pred,
                     multiclass=FALSE,
                     mappings = c("Background", "Positive"),
                     decimals=4)

print(metsOut)
```

`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 = levels(as.factor(reference)),
  decimals = 4
)
```

Arguments

`reference`	SpatRaster object of reference class codes/indices.
`predicted`	SpatRaster object of reference class codes/indices.
`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 class names. These must be in the same order as the class indices or class names so that they are correctly matched to the correct category. If no mappings are provided, then the factor levels or class indices are used by default. For a binary classification, it is assumed that the first class is "Background" and the second class is "Positive".
`decimals`	Number of decimal places to return for assessment metrics. Default is 4.

Details

This function generates a set of summary assessment metrics when provided reference and predicted classes. Results are returned as a list object. For multiclass assessment, the class names ($Classes), count of samples per class in the reference data ($referenceCounts), count of samples per class in the predictions ($predictionCounts), confusion matrix ($confusionMatrix), aggregated assessment metrics ($aggMetrics) (OA = overall accuracy, macroF1 = macro-averaged class aggregated F1-score, macroPA = macro-averaged class aggregated producer's accuracy or recall, and macroUA = macro-averaged class aggregated user's accuracy or precision), class-level user's accuracies or precisions ($userAccuracies), class-level producer's accuracies or recalls ($producerAccuracies), and class-level F1-scores ($F1Scores). For a binary case, the $Classes, $referenceCounts, $predictionCounts, and $confusionMatrix objects are also returned; however, the $aggMets object is replaced with $Mets, which stores the following metrics: overall accuracy, recall, precision, specificity, negative predictive value (NPV), and F1-score. For binary cases, the second class is assumed to be the positive case.

Value

List object containing the resulting metrics and ancillary information.

Examples

```
if(requireNamespace("terra", quietly = TRUE)){
require(torch)
require(terra)
#Multiclass example

#Generate example data as SpatRasters
ref <- terra::rast(matrix(sample(c(1, 2, 3), 625, replace=TRUE), nrow=25, ncol=25))
pred <- terra::rast(matrix(sample(c(1, 2, 3), 625, replace=TRUE), nrow=25, ncol=25))
```

```
#Calculate metrics
metsOut <- assessRaster(reference=ref,
                        predicted=pred,
                        multiclass=TRUE,
                        mappings=c("Class A", "Class B", "Class C"),
                        decimals=4)

print(metsOut)

#Binary example

#Generate example data as SpatRasters
ref <- terra::rast(matrix(sample(c(0, 1), 625, replace=TRUE), nrow=25, ncol=25))
pred <- terra::rast(matrix(sample(c(0, 1), 625, replace=TRUE), nrow=25, ncol=25))

#Calculate metrics
metsOut <- assessRaster(reference=ref,
                        predicted=pred,
                        multiclass=FALSE,
                        mappings=c("Background", "Positive"),
                        decimals=4)

print(metsOut)
}
```

`defineMobileUNet` *defineMobileUNet*

Description

Define a UNet architecture for geospatial semantic segmentation with a MobileNet-v2 backbone.

Usage

```
defineMobileUNet(
  nCls = 3,
  pretrainedEncoder = TRUE,
  freezeEncoder = TRUE,
  actFunc = "relu",
  useAttn = FALSE,
  useDS = FALSE,
  dcChn = c(256, 128, 64, 32, 16),
  negative_slope = 0.01
)
```

Arguments

`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. Default is 3.

`pretrainedEncoder`	TRUE or FALSE. Whether or not to initialized using pre-trained ImageNet weights for the MobileNet-v2 encoder. Default is TRUE.
`freezeEncoder`	TRUE or FALSE. Whether or not to freeze the encoder during training. The default is TRUE. If TRUE, only the decoder component is trained.
`actFunc`	Defines activation function to use throughout the network (note that MobileNet-v2 layers are not impacted). "relu" = rectified linear unit (ReLU); "lrelu" = leaky ReLU; "swish" = swish. Default is "relu".
`useAttn`	TRUE or FALSE. Whether to add attention gates along the skip connections. Default is FALSE or no attention gates are added.
`useDS`	TRUE or FALSE. Whether or not to use deep supervision. If TRUE, four predictions are made, one at each of the four largest decoder block resolutions, and the predictions are returned as a list object containing the 4 predictions. If FALSE, only the final prediction at the original resolution is returned. Default is FALSE or deep supervision is not implemented.
`dcChn`	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.
`negative_slope`	If actFunc = "lrelu", specifies the negative slope term to use. Default is 0.01.

Details

Define a UNet architecture with a MobileNet-v2 backbone or encoder. This UNet implementation was inspired by a blog post by Sigrid Keydana available here. This architecture has 6 blocks in the encoder (including the bottleneck) and 5 blocks in the decoder. The user is able to implement deep supervision (useDS = TRUE) and attention gates along the skip connections (useAttn = TRUE). This model requires three input bands or channels.

Value

ModileUNet model instance as torch nn_module

Examples

```
require(torch)
#Generate example data as torch tensor
tensorIn <- torch::torch_rand(c(12,3,128,128))

#Instantiate model
model <- defineMobileUNet(nCls = 3,
                          pretrainedEncoder = FALSE,
                          freezeEncoder = FALSE,
                          actFunc = "relu",
                          useAttn = TRUE,
                          useDS = TRUE,
                          dcChn = c(256,128,64,32,16),
                          negative_slope = 0.01)

pred <- model(tensorIn)
```

`defineSegDataSet` *defineSegDataSet*

Description

Instantiate a subclass of torch::dataset() for geospatial semantic segmentation

Usage

```
defineSegDataSet(
  chpDF,
  folder,
  normalize = FALSE,
  rescaleFactor = 1,
  mskRescale = 1,
  mskAdd = 0,
  bands = c(1, 2, 3),
  bMns = 1,
  bSDs = 1,
  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 and mask 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 are 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.
`mskAdd`	Value to add to mask class numeric codes. For example, if class indices start are 0, 1 can be added so that indices start at 1. Default is 0 (return original class codes). Note that several other functions in this package have a zeroStart parameter. If class codes start at 0, this argument should be set to TRUE. If they start at 1, this argument should be set to FALSE. The importance of this arises from the use of one-hot encoding internally, which requires that class indices start at 1.
`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.
`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. This is only applicable to RGB data.
`probSaturation`	0 to 1. Probability of applying saturation augmentations. Default is 0 or no augmentations. Must be changed if augmentations are desired. This is only applicable to RGB data.
`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. For example, 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. For example, 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. A gamma larger than 1 makes the shadows darker while a gamma smaller than 1 makes 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 range -0.5 to 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. For example, 0 will give a black-and-white image, 1 will give the original image, and 2 will enhance the saturation by a factor of 2.

Details

This function instantiates a subclass of torch::dataset() that loads data generated using the makeChips() or makeChipsMultiClass() function. Can also define random augmentations to combat overfitting. Note that horizontal and vertical flips will affect the alignment of the image and associated mask chips. As a result, the same augmentation will be applied to both the image and 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. Predictor variables are generated with three dimensions (channel/variable, width, height) regardless of the number of channels/variables. Masks are generated as three dimensional tensors (class index, width, height).

Value

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

Examples

```
## Not run:
#Define training dataset and augmentations
trainDS <- defineSegDataSet(
  chpDF=trainDF,
  folder="PATH TO CHIPS FOLDER",
  normalize = FALSE,
  rescaleFactor = 255,
  mskRescale= 255,
  bands = c(1,2,3),
  mskAdd=1,
  doAugs = TRUE,
  maxAugs = 1,
```

```
  probVFlip = .5,
  probHFlip = .5,
  probBrightness = 0,
  probContrast = 0,
  probGamma = 0,
  probHue = 0,
  probSaturation = 0,
  brightFactor = c(.9,1.1),
  contrastFactor = c(.9,1.1),
  gammaFactor = c(.9, 1.1, 1),
  hueFactor = c(-.1, .1),
  saturationFactor = c(.9, 1.1))

## End(Not run)
```

`defineUNet` *defineUNet*

Description

Define a UNet architecture for geospatial semantic segmentation.

Usage

```
defineUNet(
  inChn = 3,
  nCls = 3,
  actFunc = "relu",
  useAttn = FALSE,
  useSE = FALSE,
  useRes = FALSE,
  useASPP = FALSE,
  useDS = FALSE,
  enChn = c(16, 32, 64, 128),
  dcChn = c(128, 64, 32, 16),
  btnChn = 256,
  dilRates = c(1, 2, 4, 8, 16),
  dilChn = c(16, 16, 16, 16, 16),
  negative_slope = 0.01,
  seRatio = 8
)
```

Arguments

`inChn`	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. Default is 3.

`actFunc`	Defines activation function to use throughout the network. "relu" = rectified linear unit (ReLU); "lrelu" = leaky ReLU; "swish" = swish. Default is "relu".
`useAttn`	TRUE or FALSE. Whether to add attention gates along the skip connections. Default is FALSE or no attention gates are added.
`useSE`	TRUE or FALSE. Whether or not to include squeeze and excitation modules in the encoder. Default is FALSE or no squeeze and excitation modules are used.
`useRes`	TRUE or FALSE. Whether to include residual connections in the encoder, decoder, and bottleneck/ASPP module blocks. Default is FALSE or no residual connections are included.
`useASPP`	TRUE or FALSE. Whether to use an ASPP module as the bottleneck as opposed to a double convolution operation. Default is FALSE or the ASPP module is not used as the bottleneck.
`useDS`	TRUE or FALSE. Whether or not to use deep supervision. If TRUE, four predictions are made, one at each decoder block resolution, and the predictions are returned as a list object containing the 4 predictions. If FALSE, only the final prediction at the original resolution is returned. Default is FALSE or deep supervision is not implemented.
`enChn`	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.
`dcChn`	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.
`btnChn`	Number of output feature maps from the bottleneck block. Default is 256.
`dilRates`	Vector of 5 values specifying the dilation rates used in the ASPP module. Default is 1, 2, 4, 6, and 16.
`dilChn`	Vector of 5 values specifying the number of channels to produce at each dilation rate within the ASPP module. Default is 16 for each dilation rate or 80 channels overall.
`negative_slope`	If actFunc = "lrelu", specifies the negative slope term to use. Default is 0.01.
`seRatio`	Ratio to use in squeeze and excitation module. The default is 8.

Details

Define a 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. Users can also choose to (1) replace all ReLU activation functions with leaky ReLU or swish, (2) implement attention gates along the skip connections, (3) implement squeeze and excitation modules within the encoder blocks, (4) add residual connections within all blocks, (5) replace the bottleneck with a modified atrous spatial pyramid pooling (ASPP) module, and/or (6) implement deep supervision using predictions generated at each stage in the decoder.

Value

Unet model instance as torch nnn_module

Examples

```
require(torch)
# example code
#Generate example data as torch tensor
tensorIn <- torch::torch_rand(c(12,4,128,128))

 #Instantiate model
 model <- defineUNet(inChn = 4,
                    nCls = 3,
                    actFunc = "lrelu",
                    useAttn = TRUE,
                    useSE = TRUE,
                    useRes = TRUE,
                    useASPP = TRUE,
                    useDS = TRUE,
                    enChn = c(16,32,64,128),
                    dcChn = c(128,64,32,16),
                    btnChn = 256,
                    dilRates=c(1,2,4,8,16),
                    dilChn=c(16,16,16,16,16),
                    negative_slope = 0.01,
                    seRatio=8)

 #Predict data with model
 pred <- model(tensorIn)
```

`defineUnifiedFocalLoss`
defineUnifiedFocalLoss

Description

Define a loss for semantic segmentation using a modified unified focal loss framework as a subclass of torch::nn_module()

Usage

```
defineUnifiedFocalLoss(
  nCls = 3,
  lambda = 0.5,
  gamma = 0.5,
  delta = 0.6,
  smooth = 1e-08,
  zeroStart = TRUE,
  clsWghtsDist = 1,
  clsWghtsReg = 1,
  useLogCosH = FALSE,
  device = "cuda"
)
```

Arguments

`nCls`	Number of classes being differentiated.
`lambda`	Term used to control the relative weighting of the distribution- and region-based losses. Default is 0.5, or equal weighting between the losses. If lambda = 1, only the distribution- based loss is considered. If lambda = 0, only the region-based loss is considered. Values between 0.5 and 1 put more weight on the distribution-based loss while values between 0 and 0.5 put more weight on the region-based loss.
`gamma`	Parameter that controls weighting applied to difficult-to-predict pixels (for distribution-based losses) or difficult-to-predict classes (for region-based losses). Smaller values increase the weight applied to difficult samples or classes. Default is 1, or no focal weighting is applied. Value must be less than or equal to 1 and larger than 0.
`delta`	Parameter that controls the relative weightings of false positive and false negative errors for each class. Different weightings can be provided for each class. The default is 0.6, which results in prioritizing false negative errors relative to false positive errors.
`smooth`	Smoothing factor to avoid divide-by-zero errors and provide numeric stability. Default is 1e-8. Recommend using the default.
`zeroStart`	TRUE or FALSE. If class indices start at 0 as opposed to 1, this should be set to TRUE. This is required to implement one-hot encoding since R starts indexing at 1. Default is TRUE.
`clsWghtsDist`	Vector of class weights for use in calculating a weighted version of the CE loss. Default is for all classes to be equally weighted.
`clsWghtsReg`	Vector of class weights for use in calculating a weighted version of the region-based loss. Default is for all classes to be equally weighted.
`useLogCosH`	TRUE or FALSE. Whether or not to apply a logCosH transformation to the region-based loss. Default is FALSE.
`device`	Define device being used for computation. Define using torch_device().

Details

Implementation of modified version of the unified focal loss after:

Yeung, M., Sala, E., Schönlieb, C.B. and Rundo, L., 2022. Unified focal loss: Generalising Dice and cross entropy-based losses to handle class imbalanced medical image segmentation. Computerized Medical Imaging and Graphics, 95, p.102026.

Modifications include (1) allowing users to define class weights for both the distribution- based and region-based losses, (2) using class weights as opposed to the symmetric and asymmetric methods implemented by the authors, and (3) including an option to apply a logcosh transform to the region-based loss.

This loss has three key hyperparameters that control its implementation. Lambda controls the relative weight of the distribution- and region-based losses. Default is 0.5, or equal weighting between the losses is applied. If lambda = 1, only the distribution- based loss is considered. If lambda = 0, only the region-based loss is considered. Values between 0.5 and 1 put more weight on the distribution-based loss while values between 0 and 0.5 put more weight on the region-based loss.

Gamma controls the application of focal loss and the application of increased weight to difficult-to-predict pixels (for distribution-based losses) or difficult-to-predict classes (region-based losses). Lower gamma values put increased weight on difficult samples or classes. Using a value of 1 equates to not using a focal adjustment.

The delta term controls the relative weight of false positive and false negative errors for each class. The default is 0.6 for each class, which results in placing a higher weight on false negative as opposed to false positive errors relative to that class.

By adjusting the lambda, gamma, delta, and class weight terms, the user can implement a variety of different loss metrics including cross entropy loss, weighted cross entropy loss, focal cross entropy loss, focal weighted cross entropy loss, Dice loss, focal Dice loss, Tversky loss, and focal Tversky loss.

Value

Loss metric for use in training process.

Examples

```
library(terra)
library(torch)
#Generate example data as SpatRasters
ref <- terra::rast(matrix(sample(c(1, 2, 3), 625, replace=TRUE), nrow=25, ncol=25))
pred1 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred2 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred3 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred <- c(pred2, pred2, pred3)

#Convert SpatRaster to array
ref <- terra::as.array(ref)
pred <- terra::as.array(pred)

#Convert arrays to tensors and reshape
ref <- torch::torch_tensor(ref, dtype=torch::torch_long())
pred <- torch::torch_tensor(pred, dtype=torch::torch_float32())
ref <- ref$permute(c(3,1,2))
pred <- pred$permute(c(3,1,2))

#Add mini-batch dimension
ref <- ref$unsqueeze(1)
pred <- pred$unsqueeze(1)

#Duplicate tensors to have a batch of two
ref <- torch::torch_cat(list(ref, ref), dim=1)
pred <- torch::torch_cat(list(pred, pred), dim=1)

#Instantiate loss metric
myDiceLoss <- defineUnifiedFocalLoss(nCls=3,
                                    lambda=0, #Only use region-based loss
                                    gamma= 1,
                                    delta= 0.5, #Equal weights for FP and FN
                                    smooth = 1e-8,
```

```
                                zeroStart=FALSE,
                                clsWghtsDist=1,
                                clsWghtsReg=1,
                                useLogCosH =FALSE,
                                device='cpu')
#Calculate loss
myDiceLoss(pred, ref)
```

`defineUnifiedFocalLossDS` *defineUnifiedFocalLossDS*

Description

Define a loss for geospatial semantic segmentation using a modified unified focal loss framework as a subclass of torch::nn_module() when using deep supervision.

Usage

```
defineUnifiedFocalLossDS(
  nCls = 3,
  dsWghts = c(0.6, 0.2, 0.1, 0.1),
  lambda = 0.5,
  gamma = 0.5,
  delta = 0.6,
  smooth = 1e-08,
  zeroStart = TRUE,
  clsWghtsDist = 1,
  clsWghtsReg = 1,
  useLogCosH = FALSE,
  device = "cuda"
)
```

Arguments

`nCls`	Number of classes being differentiated.
`dsWghts`	Vector of 4 weights. Weights to apply to the losses calculated at each spatial resolution when using deep supervision. The default is c(.6, .2, .1, .1) where larger weights are placed on the results at a higher spatial resolution.
`lambda`	Term used to control the relative weighting of the distribution- and region-based losses. Default is 0.5, or equal weighting between the losses. If lambda = 1, only the distribution- based loss is considered. If lambda = 0, only the region-based loss is considered. Values between 0.5 and 1 put more weight on the distribution-based loss while values between 0 and 0.5 put more weight on the region-based loss.

`gamma`	Parameter that controls weighting applied to difficult-to-predict pixels (for distribution-based losses) or difficult-to-predict classes (for region-based losses). Smaller values increase the weight applied to difficult samples or classes. Default is 1, or no focal weighting is applied. Value must be less than or equal to 1 and larger than 0.
`delta`	Parameter that controls the relative weightings of false positive and false negative errors for each class. Different weightings can be provided for each class. The default is 0.6, which results in prioritizing false negative errors relative to false positive errors.
`smooth`	Smoothing factor to avoid divide-by-zero errors and provide numeric stability. Default is 1e-8. Recommend using the default.
`zeroStart`	TRUE or FALSE. If class indices start at 0 as opposed to 1, this should be set to TRUE. This is required to implement one-hot encoding since R starts indexing at 1. Default is TRUE.
`clsWghtsDist`	Vector of class weights for use in calculating a weighted version of the CE loss. Default is for all classes to be equally weighted.
`clsWghtsReg`	Vector of class weights for use in calculating a weighted version of the region-based loss. Default is for all classes to be equally weighted.
`useLogCosH`	TRUE or FALSE. Whether or not to apply a logCosH transformation to the region-based loss. Default is FALSE.
`device`	Define device being used for computation. Define using torch_device().

Details

Implementation of modified version of the unified focal loss after:

Yeung, M., Sala, E., Schönlieb, C.B. and Rundo, L., 2022. Unified focal loss: Generalising Dice and cross entropy-based losses to handle class imbalanced medical image segmentation. Computerized Medical Imaging and Graphics, 95, p.102026.

Modifications include (1) allowing users to define class weights for both the distribution- based and region-based losses, (2) using class weights as opposed to the symmetric and asymmetric methods implemented by the authors, and (3) including an option to apply a logcosh transform to the region-based loss.

This loss has three key hyperparameters that control its implementation. Lambda controls the relative weight of the distribution- and region-based losses. Default is 0.5, or equal weighting between the losses is applied. If lambda = 1, only the distribution- based loss is considered. If lambda = 0, only the region-based loss is considered. Values between 0.5 and 1 put more weight on the distribution-based loss while values between 0 and 0.5 put more weight on the region-based loss.

Gamma controls the application of focal loss and the application of increased weight to difficult-to-predict pixels (for distribution-based losses) or difficult-to-predict classes (region-based losses). Lower gamma values put increased weight on difficult samples or classes. Using a value of 1 equates to not using a focal adjustment.

The delta term controls the relative weight of false positive and false negative errors for each class. The default is 0.6 for each class, which results in placing a higher weight on false negative as opposed to false positive errors relative to that class.

By adjusting the lambda, gamma, delta, and class weight terms, the user can implement a variety of different loss metrics including cross entropy loss, weighted cross entropy loss, focal cross entropy loss, focal weighted cross entropy loss, Dice loss, focal Dice loss, Tversky loss, and focal Tversky loss.

Value

Loss metric for use in training process.

`describeBatch` *describeBatch*

Description

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

Usage

```
describeBatch(dataLoader, zeroStart = FALSE)
```

Arguments

`dataLoader`	Instantiated instance of a DataLoader created using torch::dataloader().
`zeroStart`	TRUE or FALSE. If class indices start at 0, set this to TRUE. If they start at 1, set this to FALSE. Default is FALSE.

Details

The goal of this function is to provide a check of a mini-batch of image chips and associated masks generated by a DataLoader instance using defineSegDataSet(). Summary information includes the mini-batch size (batchSize); image chip data type (imageDataType); mask data type (maskDataType); the shape of the mini-batch of images or predictor variables as mini-batch size, number of channels, width pixel count, and height pixel count (imageShape); the mask shape (maskShape); image band means (bndMns); image band standard deviations (bndSDs); count of pixels in each class in the mini-batch (maskCnts); and minimum (minIndex) and maximum (maxIndex) class indices present in the mini-batch.

Value

List object summarizing a mini-batch of image chips and masks.

Examples

```
## Not run:
trainStats <- describeBatch(trainDL,
tzeroStart=TRUE,
tusedDS=FALSE)

## End(Not run)
```

`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/or 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 generates a set of summary metrics from image chips and associated masks stored in a directory. For each band, the minimum, median, mean, maximum, and standard deviation are returned (along with some other metrics). 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.

Examples

```
## Not run:
chpDescript <- describeChips(folder= "PATH TO CHIPS FOLDER",
                             extension = ".tif",
                             mode = "Positive",
                             subSample = TRUE,
                             numChips = 100,
                             numChipsBack = 100,
                             subSamplePix = TRUE,
                             sampsPerChip = 400)

## End(Not run)
```

`luz_metric_f1score` *luz_metric_f1score*

Description

luz_metric function to calculate the macro-averaged, class aggregated F1-score

Usage

```
luz_metric_f1score(
  nCls = 1,
  smooth = 1,
  mode = "multiclass",
  biThresh = 0.5,
  clsWghts = rep(1, nCls),
  zeroStart = TRUE,
  usedDS = FALSE
)
```

Arguments

`nCls`	Number of classes being differentiated.
`smooth`	A smoothing factor to avoid divide by zero errors. Default is 1.
`mode`	Either "binary" or "multiclass". If "binary", only the logit for the 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. Note that this package is designed to treat all predictions as multiclass. The "binary" mode is only provided for use outside of the standard geodl workflow.
`biThresh`	Probability threshold to define postive case prediction. Default is 0.5.
`clsWghts`	Vector of class weightings loss calculatokn. Default is equal weightings.
`zeroStart`	TRUE or FALSE. If class indices start at 0 as opposed to 1, this should be set to TRUE. This is required to implement one-hot encoding since R starts indexing at 1. Default is TRUE.
`usedDS`	TRUE or FALSE. If deep supervision was implemented and masks are produced at varying scales using the defineSegDataSetDS() function, this should be set to TRUE. Only the original resolution is used to calculate assessment metrics. Default is FALSE.

Details

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

Value

Calculated metric returned as a base-R vector as opposed to tensor.

Examples

```
library(terra)
library(torch)
#Generate example data as SpatRasters
ref <- terra::rast(matrix(sample(c(1, 2, 3), 625, replace=TRUE), nrow=25, ncol=25))
pred1 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred2 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred3 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred <- c(pred2, pred2, pred3)

#Convert SpatRaster to array
ref <- terra::as.array(ref)
pred <- terra::as.array(pred)

#Convert arrays to tensors and reshape
ref <- torch::torch_tensor(ref, dtype=torch::torch_long())
pred <- torch::torch_tensor(pred, dtype=torch::torch_float32())
ref <- ref$permute(c(3,1,2))
pred <- pred$permute(c(3,1,2))
```

```
#Add mini-batch dimension
ref <- ref$unsqueeze(1)
pred <- pred$unsqueeze(1)

#Duplicate tensors to have a batch of two
ref <- torch::torch_cat(list(ref, ref), dim=1)
pred <- torch::torch_cat(list(pred, pred), dim=1)

#Calculate Macro-Averaged, Class Aggregated F1-Score
metric<-luz_metric_f1score(nCls=3,
                           smooth=1e-8,
                           mode = "multiclass",
                           zeroStart=FALSE,
                           usedDS=FALSE)
metric<-metric$new()
metric$update(pred,ref)
metric$compute()
```

`luz_metric_overall_accuracy`

luz_metric_overall_accuracy

Description

luz_metric function to calculate overall accuracy ((correct/total)*100)

Usage

```
luz_metric_overall_accuracy(
  nCls = 1,
  smooth = 1,
  mode = "multiclass",
  biThresh = 0.5,
  zeroStart = TRUE,
  usedDS = FALSE
)
```

Arguments

`nCls`	Number of classes being differentiated.
`smooth`	A smoothing factor to avoid divide by zero errors. Default is 1.
`mode`	Either "binary" or "multiclass". If "binary", only the logit for the 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. Note that this package is designed to treat all predictions as multiclass. The "binary" mode is only provided for use outside of the standard geodl workflow.

`biThresh`	Probability threshold to define postive case prediction. Default is 0.5.
`zeroStart`	TRUE or FALSE. If class indices start at 0 as opposed to 1, this should be set to TRUE. This is required to implement one-hot encoding since R starts indexing at 1. Default is TRUE.
`usedDS`	TRUE or FALSE. If deep supervision was implemented and masks are produced at varying scales using the defineSegDataSetDS() function, this should be set to TRUE. Only the original resolution is used to calculate assessment metrics. Default is FALSE.

Value

Calculated metric returned as a base-R vector as opposed to tensor.

Examples

```
require(terra)
require(torch)
#Generate example data as SpatRasters
ref <- terra::rast(matrix(sample(c(1, 2, 3), 625, replace=TRUE), nrow=25, ncol=25))
pred1 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred2 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred3 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred <- c(pred2, pred2, pred3)

#Convert SpatRaster to array
ref <- terra::as.array(ref)
pred <- terra::as.array(pred)

#Convert arrays to tensors and reshape
ref <- torch::torch_tensor(ref, dtype=torch::torch_long())
pred <- torch::torch_tensor(pred, dtype=torch::torch_float32())
ref <- ref$permute(c(3,1,2))
pred <- pred$permute(c(3,1,2))

#Add mini-batch dimension
ref <- ref$unsqueeze(1)
pred <- pred$unsqueeze(1)

#Duplicate tensors to have a batch of two
ref <- torch::torch_cat(list(ref, ref), dim=1)
pred <- torch::torch_cat(list(pred, pred), dim=1)

#Calculate Overall Accuracy
metric<-luz_metric_overall_accuracy(nCls=3,
                                    smooth=1e-8,
                                     mode = "multiclass",
                                    zeroStart=FALSE,
                                    usedDS=FALSE)
metric<-metric$new()
metric$update(pred,ref)
metric$compute()
```

`luz_metric_precision` *luz_metric_precision*

Description

luz_metric function to calculate macro-averaged, class aggregated precision

Usage

```
luz_metric_precision(
  nCls = 3,
  smooth = 1,
  mode = "multiclass",
  biThresh = 0.5,
  zeroStart = TRUE,
  clsWghts = rep(1, nCls),
  usedDS = TRUE
)
```

Arguments

`nCls`	Number of classes being differentiated.
`smooth`	A smoothing factor to avoid divide by zero errors. Default is 1.
`mode`	Either "binary" or "multiclass". If "binary", only the logit for the 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. Note that this package is designed to treat all predictions as multiclass. The "binary" mode is only provided for use outside of the standard geodl workflow.
`biThresh`	Probability threshold to define postive case prediction. Default is 0.5.
`zeroStart`	TRUE or FALSE. If class indices start at 0 as opposed to 1, this should be set to TRUE. This is required to implement one-hot encoding since R starts indexing at 1. Default is TRUE.
`clsWghts`	Vector of class weightings loss calculatokn. Default is equal weightings.
`usedDS`	TRUE or FALSE. If deep supervision was implemented and masks are produced at varying scales using the defineSegDataSetDS() function, this should be set to TRUE. Only the original resolution is used to calculate assessment metrics. Default is FALSE.

Details

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

Value

Calculated metric returned as a base-R vector as opposed to tensor.

Examples

```
library(terra)
library(torch)
#Generate example data as SpatRasters
ref <- terra::rast(matrix(sample(c(1, 2, 3), 625, replace=TRUE), nrow=25, ncol=25))
pred1 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred2 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred3 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred <- c(pred2, pred2, pred3)

#Convert SpatRaster to array
ref <- terra::as.array(ref)
pred <- terra::as.array(pred)

#Convert arrays to tensors and reshape
ref <- torch::torch_tensor(ref, dtype=torch::torch_long())
pred <- torch::torch_tensor(pred, dtype=torch::torch_float32())
ref <- ref$permute(c(3,1,2))
pred <- pred$permute(c(3,1,2))

#Add mini-batch dimension
ref <- ref$unsqueeze(1)
pred <- pred$unsqueeze(1)

#Duplicate tensors to have a batch of two
ref <- torch::torch_cat(list(ref, ref), dim=1)
pred <- torch::torch_cat(list(pred, pred), dim=1)
#Calculate Macro-Averaged, Class Aggregated Precision
metric<-luz_metric_precision(nCls=3,
                             smooth=1e-8,
                             mode = "multiclass",
                             zeroStart=FALSE,
                             usedDS=FALSE)
metric<-metric$new()
metric$update(pred,ref)
metric$compute()
```

`luz_metric_recall` *luz_metric_recall*

Description

luz_metric function to calculate macro-averaged, class aggregated recall

Usage

```
luz_metric_recall(
  nCls = 3,
```

```
  smooth = 1,
  mode = "multiclass",
  biThresh = 0.5,
  zeroStart = TRUE,
  clsWghts = rep(1, nCls),
  usedDS = TRUE
)
```

Arguments

`nCls`	Number of classes being differentiated.
`smooth`	A smoothing factor to avoid divide by zero errors. Default is 1.
`mode`	Either "binary" or "multiclass". If "binary", only the logit for the 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. Note that this package is designed to treat all predictions as multiclass. The "binary" mode is only provided for use outside of the standard geodl workflow.
`biThresh`	Probability threshold to define postive case prediction. Default is 0.5.
`zeroStart`	TRUE or FALSE. If class indices start at 0 as opposed to 1, this should be set to TRUE. This is required to implement one-hot encoding since R starts indexing at 1. Default is TRUE.
`clsWghts`	Vector of class weightings loss calculatokn. Default is equal weightings.
`usedDS`	TRUE or FALSE. If deep supervision was implemented and masks are produced at varying scales using the defineSegDataSetDS() function, this should be set to TRUE. Only the original resolution is used to calculate assessment metrics. Default is FALSE.

Details

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

Value

Calculated metric returned as a base-R vector as opposed to tensor.

Examples

```
library(terra)
library(torch)
#Generate example data as SpatRasters
ref <- terra::rast(matrix(sample(c(1, 2, 3), 625, replace=TRUE), nrow=25, ncol=25))
pred1 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred2 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred3 <- terra::rast(matrix(sample(c(1:150), 625, replace=TRUE), nrow=25, ncol=25))
pred <- c(pred2, pred2, pred3)

#Convert SpatRaster to array
```

```
ref <- terra::as.array(ref)
pred <- terra::as.array(pred)

#Convert arrays to tensors and reshape
ref <- torch::torch_tensor(ref, dtype=torch::torch_long())
pred <- torch::torch_tensor(pred, dtype=torch::torch_float32())
ref <- ref$permute(c(3,1,2))
pred <- pred$permute(c(3,1,2))

#Add mini-batch dimension
ref <- ref$unsqueeze(1)
pred <- pred$unsqueeze(1)

#Duplicate tensors to have a batch of two
ref <- torch::torch_cat(list(ref, ref), dim=1)
pred <- torch::torch_cat(list(pred, pred), dim=1)

#Calculate Macro-Averaged, Class Aggregated Recall
metric<-luz_metric_recall(nCls=3,
                          smooth=1e-8,
                          mode = "multiclass",
                          zeroStart=FALSE,
                          usedDS=FALSE)
metric<-metric$new()
metric$update(pred,ref)
metric$compute()
```

`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",
  useExistingDir = FALSE
)
```

Arguments

`image`	SpatRaster object or 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`	SpatRaster object or 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.
`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 if useExistingDir = FALSE. 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".
`useExistingDir`	TRUE or FALSE. Write chips into an existing directory with subfolders already defined as opposed to using a new directory. This can be used if you want to add chips to an existing set of chips. However, the "mode" should be the same as that used to generated the original chips. Default is FALSE.

Details

This function generates 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 are 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. No R object is returned.

Examples

```
## Not run:
makeChips(image = "INPUT IMAGE FILE NAME AND PATH",
          mask = "INPUT RASTER MASK FILE NAME AND PATH",
          n_channels = 3,
          size = 256,
          stride_x = 256,
          stride_y = 256,
          outDir = "OUTPUT DIRECTY IN WHICH TO SAVE CHIPS",
          mode = "Positive",
          useExistingDir=FALSE)

## End(Not run)
```

`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 (chpN, chpPth, and mskPth) and, optionally, a CSV file written to disk. If mode = "Divided", a division column is added to differentiate "positive" and "background" samples.

Examples

```
## Not run:
chpDF <- makeChipsDF(folder = "PATHT TO CHIPS FOLDER",
                     outCSV = "OUTPUT CSV FILE AND PATH",
                     extension = ".tif",
                     mode="Positive",
                     shuffle=TRUE,
                     saveCSV=TRUE)

## End(Not run)
```

`makeChipsMultiClass` *makeChipsMultiClass*

Description

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

Usage

```
makeChipsMultiClass(
  image,
  mask,
  n_channels = 3,
  size = 256,
```

```
  stride_x = 256,
  stride_y = 256,
  outDir,
  useExistingDir = FALSE
)
```

Arguments

`image`	SpatRaster object or 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`	SpatRaster object or 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.
`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 if useExistingDir = FALSE. You must include the final forward slash in the file path (e.g., "C:/data/chips/").
`useExistingDir`	TRUE or FALSE. Write chips into an existing directory with subfolders already defined as opposed to using a new directory. This can be used if you want to add chips to an existing set of chips. Default is FALSE.

Details

This function generates 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 or when only two classes are differentiated, 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.

Examples

```
## Not run:
makeChipsMultiClass(image = "INPUT IMAGE NAME AND PATH",
                    mask = "INPUT RASTER MASK NAME AND PATH",
                    n_channels = 3,
                    size = 512,
                    stride_x = 512,
                    stride_y = 512,
                    outDir = "DIRECTORY IN WHICH TO SAVE CHIPS",
                    useExistingDir=FALSE)

## End(Not run)
```

`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. 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 differentiates 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 1 to n. If no cropping is applied, the generated raster mask will 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.

Examples

```
## Not run:
makeMasks(image = "INPUT IMAGE FILE AND PATH",
          features = "INPUT VECTOR FEATURES FILE AND PATH",
          crop = TRUE,
          extent = "INPUT VECTOR BOUNDARY and PATH",
          field = "ATTRIBUTE COLUMN NAME",
          background = 0,
```

```
          outImage = "OUTPUT IMAGE NAME AND PATH",
          outMask = "OUTPUT MASK NAME AND PATH",
          mode = "Both")

## End(Not run)
```

`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 and was originally proposed by William Odom of the United States Geological Survey (USGS):

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 grid written to disk in TIFF format and spatRaster object.

Examples

```
## Not run:
inDTM <- terra::rast("INPUT DTM NAME AND PATH")
terrOut <- makeTerrainDerivatives(dtm=inDTM,
                                  res=2,
                                  filename="OUT TERRAIN STACK NAME AND PATH")

## End(Not run)
```

`predictSpatial` *predictSpatial*

Description

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

Usage

```
predictSpatial(
  imgIn,
  model,
  predOut,
  mode = "multiclass",
  predType = "class",
  biThresh = 0.5,
  useCUDA = FALSE,
  nCls,
  chpSize,
  stride_x,
  stride_y,
  crop,
  nChn = 3,
  normalize = FALSE,
  bMns,
  bSDs,
  rescaleFactor = 1,
  usedDS = FALSE
)
```

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, should be "binary". If two or more class logits are returned, this should be "multiclass". This package treats all cases as multiclass.
`predType`	"class", "logit", or "prob". Default is "class". Whether to generate a "hard" classification ("class"), logit(s) ("logit"), or rescaled logit(s) ("prob") with a sigmoid or softmax activation applied for the positive class or each predicted class. If "class", a single-band raster of class indices is returned. If "logit" or "prob", the positive class probability is returned as a single-band raster grid for a binary classification. If "logit" or "prob" for a multiclass problem, a multiband raster grid is returned with a channel for each class.
`biThresh`	When mode = "binary" and predType = "class", threshold to use to indicate the presence class. This threshold is applied to the rescaled logits after a sigmoid activation is applied. Default is 0.5. If the rescaled logit is greater than or equal to this threshold, pixel will be mapped to the positive case. Otherwise, it will be mapped to the negative or background class. This parameter is ignored for multiclass classification.
`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 where only the positive case logit is predicted or the number of classes being differentiated 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() when model was trained.
`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().
`usedDS`	TRUE or FALSE. If model is configured to use deep supervision, this must be set to TRUE. Default is FALSE, or it is assumed that deep supervision is not used.

Details

This function generates a pixel-by-pixel prediction using input data and a trained semantic segmentation model. Can return either hard classifications, logits, or rescaled logits with a sigmoid or softmax activation applied. Note that this package treats all problems as multiclass problems and does not differentiate between binary and multiclass classification. As a result, in our workflow the multiclass mode should be used. Result is written to disk and provided as a spatRaster object. If you are experiencing memory issues, you may need to break larger raster extents into smaller tiles for processing.

Value

A spatRast object and a raster grid saved to disk of predicted class indices (predType = "class"), logits (predType = "logit"), or rescaled logits (predType = "prob").

Examples

```
## Not run:
#Predict classification
predCls <- predictSpatial(imgIn="INPUT IMAGE NAME AND PATH",
                          model=model,
                          predOut="OUTPUT RASTER NAME AND PATH",
                          mode="multiclass",
                          predType="class",
                          useCUDA=TRUE,
                          nCls=2,
                          chpSize=256,
                          stride_x=128,
                          stride_y=128,
                          crop=50,
                          nChn=3,
                          normalize=FALSE,
                          rescaleFactor=255,
                          usedDS=FALSE)

## End(Not run)
```

`viewBatch` *viewBatch*

Description

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

Usage

```
viewBatch(dataLoader, nCols = 3, r = 1, g = 2, b = 3, cNames, cColors)
```

Arguments

`dataLoader`	Instantiated instance of a DataLoader created using torch::dataloader().
`nCols`	Number of columns in the image grid. Default is 3.
`r`	Index of channel to assign to red channel. Default is 1 or first channel. 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 or the second channel. 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 or the third channel. 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 mini-batch of image chips and associated masks generated from a DataLoader.

Value

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

Examples

```
## Not run:
viewBatch(dataLoader=trainDL,
          nCols = 5,
          r = 1,
          g = 2,
          b = 3,
          cNames=c("Background", "Mine"),
          cColors=c("gray", "darksalmon"))

## End(Not run)
```

`viewBatchPreds` *viewBatchPreds*

Description

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

Usage

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

Arguments

`dataLoader`	Instantiated instance of a DataLoader created using torch::dataloader().
`model`	Fitted model used to predict mini-batch.
`mode`	"multiclass" or "binary". If the prediction returns the positive case logit for a binary classification problem, use "binary". If 2 or more class logits are returned, use "multiclass". This package treats all cases as multiclass.
`nCols`	Number of columns in the image grid. Default is 3.
`r`	Index of channel to assign to red channel. Default is 1 or the first channel. 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 or the second channel. 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 or the third channel. For gray scale or single-band images, assign the same index to all three bands.
`cCodes`	Integer codes assigned to each class. Should be in the same order as cNames.
`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 mini-batch. If FALSE, predictions will be made on the CPU. We recommend using a GPU.
`probs`	TRUE or FALSE. Default is FALSE. If TRUE, rescaled logits will be shown as opposed to the hard classification. If FALSE, hard classification will be shown. For a binary problem where only the positive case logit is returned, the logit is transformed using a sigmoid function. When 2 or more classes are predicted, softmax is used to rescale the logits.
`usedDS`	TRUE or FALSE. Must be set to TRUE when using deep supervision. Default is FALSE, or it is assumed that deep supervision is not used.

Details

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

Value

Image grids of example chips, reference masks, and predictions loaded from a mini-batch provided by the DataLoader.

Examples

```
## Not run:
viewBatchPreds(dataLoader=testDL,
               model=model,
               mode="multiclass",
               nCols =5,
               r = 1,
               g = 2,
               b = 3,
               cCodes=c(1,2),
               cNames=c("Not Mine", "Mine"),
               cColors=c("gray", "darksalmon"),
               useCUDA=TRUE,
               probs=FALSE,
               usedDS=FALSE)

## End(Not run)
```

`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", grids are produced for both the image chips and masks. Default is "both".
`justPositive`	TRUE or FALSE. If makeChips() was executed using the "Divided" mode, 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".
`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 or the first channel. For gray scale or single-band images, assign the same index to all three bands.
`g`	Band number to map to the green channel. Default is 2 or the second channel. For gray scale or single-band images, assign the same index to all three bands.
`b`	Band number to map to the red channel. Default is 3 or the third channel. For gray scale or single-band images, assign the same index to all three bands.
`rescale`	TRUE or FALSE. Whether or not to rescale image data. Default is FALSE or no rescaling.
`rescaleVal`	If rescale is TRUE, value used to rescale data. For example, 255 could be used to rescale the chips from 0 to 1 to 0 to 255.
`cNames`	Vector of class names. Class names must be provided.
`cColors`	Vector of colors (named colors, hex codes, or rgb()). Color used to visualize each class is 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 (i.e., obtain the same samples). If FALSE, seed is ignored. Default is FALSE.
`seed`	Random seed value. Default is 42. This is ignored if useSeed is FALSE.

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 makeChipsMultiClass() function. It can be 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").

Examples

```
## Not run:
viewChips(chpDF=chpDF,
          folder= "FOLDER CONTAINING CHIPS",
          nSamps = 16,
          mode = "both",
          justPositive = FALSE,
          cCnt = 4,
          rCnt = 4,
          r = 1,
          g = 2,
          b = 3,
          rescale = FALSE,
          rescaleVal = 1,
          cNames=c("Background",
                   "Building",
                   "Woodland",
                   "Water",
                   "Road"),
          cColor=c("gray",
                   "darksalmon",
                    "forestgreen",
                    "lightblue",
                    "black"),
          useSeed = FALSE,
          seed = 42)

## End(Not run)
```

Index