Package 'geodl'

November 17, 2026

Type Package

Title Geospatial Semantic Segmentation with Torch and Terra

Version 0.4.0

Date 2026-11-16

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 (>= 4.2)

Imports torch, luz, torchvision, dplyr, terra, MultiscaleDTM, psych, coro, R6, readr, rlang, sf

License GPL (>= 3)

URL https://github.com/maxwell-geospatial/geodl, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0315127, https://wvview.org/gslr/index.html

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

NeedsCompilation no

Encoding UTF-8

Suggests knitr, rmarkdown

Config/roxygen2/version 8.1.0

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

Repository CRAN

Date/Publication 2026-11-17 08:30:02 UTC

Contents

addCoordNoise . . . 3
assessDL . . . 4
assessPnts . . . 6
assessRaster . . . 8
callback_ds_weight_decay . . . 10
callback_ema . . . 11
callback_freeze_stages . . . 12
callback_per_stage_optimizer . . . 13
callback_reconfigure_loss . . . 14
callback_save_model_state_dict . . . 16
callback_training_log . . . 16
callback_unfreeze_stages . . . 17
checkDynamicChips . . . 18
countParams . . . 19
defineDLab . . . 20
defineDynamicSegDataSet . . . 22
defineEfficientUNetB2 . . . 25
defineMobileUNet . . . 27
defineRegressionLoss . . . 28
defineResNetUNet . . . 30
defineSegDataSet . . . 32
defineTerrainSeg . . . 35
defineUNet . . . 37

defineUNet3p . . . 40
defineUnifiedFocalLoss . . . 42
defineUnifiedFocalLossDS . . . 46
describeBatch . . . 49
describeChips . . . 50
luz_metric_f1score . . . 51
luz_metric_overall_accuracy . . . 53
luz_metric_precision . . . 55
luz_metric_recall . . . 57
makeAspect . . . 59
makeBoundaryDistance . . . 60
makeChips . . . 61
makeChipsDF . . . 63
makeChipsMultiClass . . . 65
makeCrv . . . 66
makeDynamicChipsSF . . . 68
makeHillshade . . . 69
makeMasks . . . 71
makeSlope . . . 73
makeTerrainVisTerra . . . 74
makeTerrainVisTorch . . . 75
makeTPI . . . 77
makeTRI . . . 78
make_ds_weights . . . 79
plotTrainingSchedule . . . 80
predictSpatial . . . 81
saveDynamicChips . . . 84
setSeed . . . 85
viewBatch . . . 86
viewBatchPreds . . . 87
viewChips . . . 89
viewDist . . . 91

Index **94**

`addCoordNoise` *addCoordNoise*

Description

Add random positional noise to point geometries.

Usage

```
addCoordNoise(
  x,
  amount,
  method = c("disc", "uniform", "gaussian"),
  seed = NULL
)
```

Arguments

x	An sf object containing only POINT geometries.
amount	Numeric scalar controlling the noise magnitude, in the units of x's CRS (e.g. meters for a projected CRS). Interpreted as the disc radius for method = "disc", the half-width of the uniform range for method = "uniform", and the standard deviation for method = "gaussian".
method	Character string specifying the noise model: "disc" (default; uniform over a circular area), "uniform" (independent uniform noise on X and Y), or "gaussian" (independent Gaussian noise on X and Y).
seed	Optional integer to seed the random number generator for reproducible noise. Default NULL (no seeding).

Details

Perturbs the coordinates of an sf POINT object by a random offset drawn from one of three spatial noise models: a uniform disc (uniform over the area of a circle of radius amount), independent uniform noise on each axis (uniform over a square of half-width amount), or independent Gaussian noise on each axis with standard deviation amount.

Value

An sf object with the same attributes as x but with perturbed point coordinates.

---

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,
  cropFactorMsk = 0,
  cropFactorPred = 0,
  usedDS = FALSE,
  useCUDA = FALSE,
  decimals = 4
)
```

Arguments

dl	torch DataLoader object.
model	instantiated model object as nn_module subclass as opposed to luz fitted object. Make sure to place model in evaluation mode.
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	mini-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".
cropFactorMsk	Number of rows and columns of cells to not include in assessment to minimize edge effects for reference mask. Default is 0 or no cropping.
cropFactorPred	Number of rows and columns of cells to not include in assessment to minimize edge effects for prediction. Default is 0 or no cropping.
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)),
  aliases = NULL,
  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".
aliases	Optional vector of display names used to label the output (confusion matrix row/column names, class name lists, and metric name vectors). Must be the same length as mappings and provided in the same order. If NULL (default), the mappings values are used as labels.
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

```
#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)

#Multiclass example with aliases to rename output labels
inDF2 <- data.frame(ref  = sample(c("cls_a", "cls_b", "cls_c"), 1000, replace=TRUE),
                    pred = sample(c("cls_a", "cls_b", "cls_c"), 1000, replace=TRUE))

metsOut2 <- assessPnts(reference=inDF2$ref,
                       predicted=inDF2$pred,
                       multiclass=TRUE,
                       mappings = c("cls_a", "cls_b", "cls_c"),
                       aliases  = c("Class A", "Class B", "Class C"),
                       decimals=4)

print(metsOut2)

#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)
}
```

`callback_ds_weight_decay` *callback_ds_weight_decay*

Description

Decay ancillary deep supervision loss weights during luz training.

Usage

```
callback_ds_weight_decay(weight_env, decay = "linear", primary_only_frac = 0.1)
```

Arguments

`weight_env`	A weight environment created by `make_ds_weights` and shared with `ds_loss`.
`decay`	Character string, either `"linear"` (default) or `"cosine"`.
`primary_only_frac`	Numeric value in `(0, 1)` specifying the fraction of total epochs at the end of training during which only the primary loss is used (ancillary weights are zero). Defaults to `0.1`.

Details

A luz callback that reduces the weights of ancillary outputs (all elements of `weight_env$values` except the first) toward zero over the decaying portion of training, leaving the primary output weight unchanged. The fraction of epochs reserved for primary-loss-only training is controlled by `primary_only_frac` (default `0.1`, i.e. the final 10\ epochs).

Two decay schedules are supported:

`"linear"` Ancillary weights decrease linearly from their initial values to zero. Matches the nnU-Net strategy.

`"cosine"` Ancillary weights follow a cosine curve, decaying slowly at first and more rapidly through mid-training.

Value

A luz callback object.

`callback_ema` *callback_ema*

Description

Maintain an exponential moving average (EMA) of model weights during training.

Usage

```
callback_ema(
  decay = 0.999,
  warmup_steps = 0L,
  save_dir = NULL,
  prefix = "ema_epoch"
)
```

Arguments

`decay`	Numeric scalar in `(0, 1)` controlling the smoothing strength. Higher values (e.g. `0.999`) produce a slower-moving average; lower values (e.g. `0.99`) track the training weights more closely. Default `0.999`.
`warmup_steps`	Non-negative integer giving the number of optimizer steps (batches) at the start of training during which the shadow weights track the training weights exactly before EMA decay begins. Default `0L`.
`save_dir`	Optional character string path to a directory. When supplied, the EMA state dictionary is saved here at the end of every epoch as a `.pt` file. Default `NULL` (no saving).
`prefix`	File-name prefix used when `save_dir` is set. Files are named `<prefix>_NNN.pt` where `NNN` is the zero-padded epoch number. Default `"ema_epoch"`.

Details

A luz callback that keeps a shadow copy of all model parameters and updates them as an exponential moving average of the training weights after each optimizer step. The EMA weights are swapped into the model during validation so that validation metrics reflect EMA performance, then the training weights are restored so training continues normally.

When `save_dir` is supplied, a complete model state dictionary with the EMA parameter values merged in is written to disk at the end of each epoch. The saved file is fully compatible with `model$load_state_dict(torch::torch_load(path))` and can be used directly for inference without any additional steps. Non-parameter buffers such as batch normalisation running statistics are taken from the live model and are not averaged, which is the standard EMA practice.

The update rule applied after each training batch is:

$$\hat{\theta} \leftarrow \text{decay} \times \hat{\theta} + (1 - \text{decay}) \times \theta$$

where $\hat{\theta}$ are the shadow (EMA) weights and θ are the current training weights.

During `warmup_steps` the shadow weights are set equal to the current training weights at every batch rather than applying the EMA rule. This prevents early random-initialisation values from being mixed into the average.

Value

A luz callback object.

`callback_freeze_stages` *callback_freeze_stages*

Description

Progressively freeze model stage parameters during training.

Usage

```
callback_freeze_stages(schedule)
```

Arguments

`schedule`	A list of named lists, each specifying one freeze action. Each entry must contain:
	`epoch` Single integer. The training epoch after which the listed components are frozen.
	`components` Character vector of submodule names to freeze at this epoch.

Details

A luz callback that sets `requires_grad = FALSE` on the parameters of specified model components at defined epoch boundaries. Multiple freeze actions at different epochs are supported via the `schedule` argument. Once frozen, a component's parameters receive no gradient and are not updated by the optimizer for the remainder of training.

Component names must match the submodule field names used inside the model. For `defineUNet` and `defineUNet3p` the stage names are `"e1"`, `"e2"`, `"e3"`, `"e4"`, `"btn"`, `"d1"`, `"d2"`, `"d3"`, `"d4"`. For `defineMobileUNet` and `defineEfficientUNetB2` there are five encoder stages (`"e1"` through

"e5") and five decoder stages ("d1" through "d5"). Attention gate and squeeze-and-excitation submodules ("ag1", "se1", etc.) can also be named.

Freezing takes effect at the **end** of the specified epoch, so frozen components are inactive from the following epoch onward. A message is printed each time a component is frozen, reporting the component name and parameter count.

Value

A luz callback object.

Examples

```
## Not run:
# Freeze early encoder stages after epoch 5,
# then the remaining encoder and bottleneck after epoch 10
callback_freeze_stages(
  schedule = list(
    list(epoch = 5,  components = c("e1", "e2")),
    list(epoch = 10, components = c("e3", "e4", "btn"))
  )
)

## End(Not run)
```

---

callback_per_stage_optimizer
callback_per_stage_optimizer

Description

Replace the luz optimizer with per-stage parameter groups at the start of training.

Usage

```
callback_per_stage_optimizer(optimizer_fn = torch::optim_adamw, ...)
```

Arguments

optimizer_fn	Optimizer constructor function. Defaults to torch::optim_adamw. Any torch optimizer that accepts a list of parameter groups as its first argument can be used.
...	Additional arguments forwarded to optimizer_fn, such as weight_decay.

Details

A luz callback that calls `model$get_param_groups()` on the instantiated model at the start of `fit()` and replaces the default flat-parameter optimizer with one built from the per-stage parameter groups. This allows different learning rates to be assigned to different stages of the network when using models that support `stageLRs` (e.g. `defineUNet`, `defineMobileUNet`, `defineUNet3p`, `defineEfficientUNetB2`).

This callback must be placed **before** any learning rate scheduler callback in the `callbacks` list so that the scheduler is initialised against the per-stage optimizer rather than the original flat one.

When using `torch::lr_one_cycle` alongside this callback, pass `max_lr`, `max_momentum`, and `base_momentum` as vectors of length equal to the number of stages so that the scheduler can assign per-group values. A proportionally scaled `max_lr` vector can be constructed as `stage_lrs / max(stage_lrs) * global_max_lr`.

Value

A luz callback object.

`callback_reconfigure_loss`
callback_reconfigure_loss

Description

Reconfigure the unified focal loss hyperparameters at specified epochs during training.

Usage

```
callback_reconfigure_loss(loss, schedule)
```

Arguments

`loss`	An instantiated `defineUnifiedFocalLoss` or `defineUnifiedFocalLossDS` object. Must be the same object passed as `loss` to `luz::setup()`.
`schedule`	A list of named lists, each specifying one reconfiguration event. Each entry must contain: `epoch` Single integer. The training epoch at the end of which the parameters are updated. The new values take effect from the following epoch onward. `params` Named list of parameter name–value pairs to update. Names must match those listed in the Details section above.

Details

A luz callback that updates one or more parameters of an instantiated `defineUnifiedFocalLoss` or `defineUnifiedFocalLossDS` loss object at defined epoch boundaries. Because torch `nn_module` objects are reference types, the same object passed here is used by the luz training loop, so changes applied by the callback take effect immediately for the following epoch.

For `defineUnifiedFocalLossDS`, parameters are routed to the inner `defineUnifiedFocalLoss` sub-object automatically.

Parameters that have pre-computed tensor counterparts (see Details) have both the scalar field and the corresponding tensor updated atomically so that the forward pass always reads consistent values.

The following parameters may be reconfigured:

`lambda` Relative weight of distribution vs. region loss.

`gamma` Focal weighting strength.

`delta` False positive / false negative balance (scalar or per-class vector).

`clsWghtsDist` Class weights for the distribution loss (scalar or per-class vector).

`clsWghtsReg` Class weights for the region loss (scalar or per-class vector).

`smooth` Smoothing constant for numeric stability.

`distThreshold` Distance threshold for edge weighting.

`minWght` Minimum per-pixel edge weight.

`maxWght` Maximum per-pixel edge weight.

`useLogCosH` Toggle log-cosh transform on the region loss.

`labelSmoothing` Label smoothing factor for the distribution loss.

`classAvgDist` Switch between macro-averaged and pixel-level distribution loss.

`prioritizeEdge` Flip which pixels receive the higher edge weight.

`edgeMode` Edge weighting mode: `"threshold"`, `"linear"`, or `"gaussian"`.

`applyDistWght` Which loss component receives edge weights: `"distribution"`, `"region"`, or `"both"`.

`cropFactorMsk` Mask crop factor.

`cropFactorPred` Prediction crop factor.

`zeroStart` Whether class indices start at 0.

Value

A luz callback object.

Examples

```
## Not run:
loss_fn <- defineUnifiedFocalLoss(nCls = 3, lambda = 0.5, gamma = 0.5, delta = 0.6)

reconfigure_cb <- callback_reconfigure_loss(
  loss = loss_fn,
  schedule = list(
```

```
    list(epoch = 5,  params = list(lambda = 0.7, gamma = 0.3)),
    list(epoch = 10, params = list(lambda = 0.9, clsWghtsDist = c(1, 2, 3))),
    list(epoch = 15, params = list(labelSmoothing = 0.1))
  )
)

fitted <- model |>
  luz::setup(loss = loss_fn, optimizer = torch::optim_adamw) |>
  luz::fit(train_dl, epochs = 20,
           callbacks = list(reconfigure_cb))

## End(Not run)
```

`callback_save_model_state_dict` *callback_save_model_state_dict*

Description

Save the model state dictionary to disk at the end of each epoch as a .pt file. For use within luz training loop.

Usage

```
callback_save_model_state_dict(save_dir = ".", prefix = "model_epoch")
```

Arguments

`save_dir`	path and directory to save state dictionary files
`prefix`	prefix for file name. Default is "model_epoch". The suffix is the epoch number.

Value

No R object is returned. The state dictionary is saved to disk as a .pt file.

`callback_training_log` *callback_training_log*

Description

Log per-epoch learning rates and deep supervision weights to a CSV file.

Usage

```
callback_training_log(path, weight_env = NULL)
```

Arguments

path	File path for the output CSV. The parent directory is created if it does not exist. Any existing file at this path is overwritten.
weight_env	Optional. A weight environment created by make_ds_weights. Supply when using deep supervision so that the current weights are logged each epoch. Omit or pass NULL when not using deep supervision.

Details

A luz callback that writes one row per training epoch recording the current optimizer learning rate(s) and, optionally, the deep supervision loss weights. Column structure is determined automatically on the first epoch:

- If the model uses per-stage learning rates via get_param_groups, each stage is logged in its own column (lr_1, lr_2, ...).
- If a single learning rate is used, a single lr column is written.
- If weight_env is supplied, columns ds_primary, ds_aux1, ds_aux2, ... are appended for each weight.

Value

A luz callback object.

---

callback_unfreeze_stages *callback_unfreeze_stages*

Description

Progressively unfreeze previously frozen model stage parameters during training.

Usage

```
callback_unfreeze_stages(schedule)
```

Arguments

schedule	A list of named lists, each specifying one unfreeze action. Each entry must contain: epoch Single integer. The training epoch after which the listed components are unfrozen. components Character vector of submodule names to unfreeze at this epoch.

Details

A luz callback that sets `requires_grad = TRUE` on the parameters of specified model components at defined epoch boundaries, reversing an earlier freeze applied by `callback_freeze_stages` or by pre-training setup (e.g. `freezeEncoder = TRUE`). Multiple unfreeze actions at different epochs are supported via the `schedule` argument.

Component names must match the submodule field names used inside the model. For `defineUNet` and `defineUNet3p` the stage names are `"e1"`, `"e2"`, `"e3"`, `"e4"`, `"btn"`, `"d1"`, `"d2"`, `"d3"`, `"d4"`. For `defineMobileUNet` and `defineEfficientUNetB2` there are five encoder stages (`"e1"` through `"e5"`) and five decoder stages (`"d1"` through `"d5"`).

Unfreezing takes effect at the **end** of the specified epoch, so unfrozen components begin receiving gradient updates from the following epoch onward. A message is printed each time a component is unfrozen, reporting the component name and parameter count. The optimizer state for the newly unfrozen parameters is not reset, so any accumulated momentum carries over.

`callback_unfreeze_stages` and `callback_freeze_stages` can be used together to implement warm-up schedules where the encoder is initially frozen while the decoder trains, then progressively unfrozen.

Value

A luz callback object.

Examples

```
## Not run:
# Train with encoder frozen for 5 epochs, then unfreeze progressively
callback_unfreeze_stages(
  schedule = list(
    list(epoch = 5,  components = c("e4", "e5")),
    list(epoch = 10, components = c("e2", "e3")),
    list(epoch = 15, components = c("e1"))
  )
)

## End(Not run)
```

`checkDynamicChips` *checkDymamicChips*

Description

Generate a dataframe of summary infformation for each chip defined using makeDynamicChipsSF().

Usage

```
checkDynamicChips(chipsSF, chipSize = 512, cellSize = 1)
```

Arguments

chipsSF	sf object created by makeDynamicChipsSF().
chipSize	Size of desired image chips. Default is 512 (512x512 cells)
cellSize	Cells size of input data. Default is 1 m.

Details

Generate set of summary information about each chip defined by makeDynamicChipsSF() including all the information in the sf object generated by makeDynamicChipsSF() and the number of columns of cells in the predictor variable raster and mask ("cCntImg" and "cCntMsk"), number of rows of cells in the predictor variable raster and mask ("rCntImg" and "rCntMsk"), number of input predictor variables ("bCntImg"), and count of NA pixels in the predictor variable raster and mask ("naCntImg" and "naCntMsk"). This function is used primarily to check the chips prior to attempting to use them in a training or validation process.

Value

Dataframe of summary information about image chips.

---

countParams *countParams*

Description

Count the number of trainable parameters in a model.

Usage

```
countParams(model, test_data)
```

Arguments

model	instantiated model object as nn_module subclass as opposed to luz fitted object.
test_data	Example tensor of the correct shape for the model being explored.

Details

Count the number of trainable parameters in an instantiated model subclassed from nn_module().

Value

Counts of model parameters.

defineDLab *defineDLab*

Description

Define a DeepLabv3+-style encoder-ASPP-decoder architecture for geospatial semantic segmentation.

Usage

```
defineDLab(
  inChn = 3,
  nCls = 3,
  actFunc = "relu",
  encBlocks = "double",
  decBlock = "double",
  useRes = FALSE,
  enChn = c(16, 32, 64, 128),
  btnChn = 256,
  lowLevelChn = 48,
  dcChn = 256,
  dilRates = c(6, 12, 18),
  dilChn = c(256, 256, 256, 256),
  negative_slope = 0.01,
  expRatio = 4
)
```

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 for the non-ConvNeXt blocks. "relu" = rectified linear unit (ReLU); "lrelu" = leaky ReLU; "swish" = swish; "gelu" = GELU. Default is "relu".
encBlocks	Block implementation for the four encoder blocks. Either a single value applied to all four blocks or a length-4 character vector, with entries "double", "dws", or "convnext". Default is "double".
decBlock	Block implementation for the decoder fusion block: "double", "dws", or "convnext". Default is "double".
useRes	TRUE or FALSE. Whether to include residual connections in the "double" and "dws" encoder and decoder blocks. Has no effect on "convnext" blocks (always residual). Default is FALSE.

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.
btnChn	Number of output feature maps from the ASPP module. Default is 256.
lowLevelChn	Number of channels to project the low-level encoder features (from the third encoder block at stride 4) to before fusing with the upsampled ASPP output. Default is 48.
dcChn	Number of output feature maps from the decoder convolution block. Default is 256.
dilRates	Vector of 3 values specifying the dilation rates used in the ASPP module. Default is 6, 12, and 18.
dilChn	Vector of 4 values specifying the number of channels to produce at each dilation rate within the ASPP module. Default is 256 for each.
negative_slope	If actFunc = "lrelu", specifies the negative slope term. Default is 0.01.
expRatio	Inverted-bottleneck expansion ratio for "convnext" blocks. Default is 4.

Details

Define an architecture with a custom 4-block encoder, an atrous spatial pyramid pooling (ASPP) module, and a lightweight decoder that fuses high-level ASPP features with low-level encoder features. The decoder uses a single skip connection from the third encoder block (stride 4) and upsamples the ASPP output (stride 16) 4x before fusion, then upsamples 4x again to reach the original resolution. This is only loosely based on DeepLabv3+: the encoder is a plain convolutional network downsampled with max pooling rather than an atrous backbone, so the name is deliberately generic.

Each of the four encoder blocks and the single decoder block can independently use one of three block implementations via `encBlocks` and `decBlock`:

- `"double"` - the standard double 3x3 convolution block (default). Becomes a residual double convolution block when `useRes = TRUE`.
- `"dws"` - a double convolution block whose second 3x3 convolution is replaced with a depthwise separable convolution (`dwsDoubleConv`). Becomes a residual variant (`dwsDoubleConvR`) when `useRes = TRUE`.
- `"convnext"` - a ConvNeXt-style block (`doubleConvBlkCNX`); always residual and always GELU, so `useRes` and `actFunc` do not affect these blocks.

The overall encoder-ASPP-decoder structure is inspired by:

Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F. and Adam, H., 2018. Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 801-818).

Value

DLab model instance as torch nn_module

defineDynamicSegDataSet *defineDynamicSegDataSet*

Description

Instantiate a subclass of torch::dataset() for geospatial semantic segmentation using dynamically generated chips

Usage

```
defineDynamicSegDataSet(
  chipsSF,
  chipSize = 512,
  cellSize = 2,
  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,
  probRotate = 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)
)

defineDynmamicSegDataSet(
  chipsSF,
  chipSize = 512,
  cellSize = 2,
  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,
  probRotate = 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

chipsSF	sf object created by makeDynamicChipsSF().
chipSize	Size of desired image chips. Default is 512 (512x512 cells)
cellSize	Cells size of input data. Default is 1 m.
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.
probRotate	0 to 1. Probability of applying rotation by 0-, 90-, 180- or 270-degrees. 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 dynmaically generates chips using the output from makeDynamicChipsSF(). 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).

`defineDynmamicSegDataSet()` is a misspelled alias retained for backward compatibility. Use `defineDynamicSegDataSet()` instead.

Value

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

---

defineEfficientUNetB2 *defineEfficientUNetB2*

Description

Define a UNet architecture for geospatial semantic segmentation with an EfficientNet-B2 backbone.

Usage

```
defineEfficientUNetB2(
  inChn = 3,
  nCls = 3,
  pretrainedEncoder = TRUE,
  freezeEncoder = FALSE,
  avgImNetWeights = FALSE,
  actFunc = "relu",
  useAttn = FALSE,
  useASPP = FALSE,
  useDS = FALSE,
  dcChn = c(256, 128, 64, 32, 16),
  btnChn = 256,
  dilRates = c(6, 12, 18),
  dilChn = c(256, 256, 256, 256),
  negative_slope = 0.01,
  stageLRs = NULL
)
```

Arguments

inChn	Number of input channels or predictor variables. 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.
pretrainedEncoder	TRUE or FALSE. Whether or not to initialize using pre-trained ImageNet weights for the EfficientNet-B2 encoder. Default is TRUE.
freezeEncoder	TRUE or FALSE. Whether or not to freeze the encoder during training. The default is FALSE. If TRUE, only the decoder component is trained.
avgImNetWeights	TRUE or FALSE. If three predictor variables are provided and ImageNet weights are used, whether or not to use the original weights or average them. If the input has more or fewer than 3 channels, the stem weights are always averaged regardless of this argument. Default is FALSE.
actFunc	Defines the activation function used in the decoder and bottleneck (note that EfficientNet-B2 encoder layers are not impacted; they use SiLU internally). "relu" = rectified linear unit (ReLU); "lrelu" = leaky ReLU; "swish" = SiLU/swish; "gelu" = GELU. Default is "relu".
useAttn	TRUE or FALSE. Whether to add attention gates along the skip connections. Default is FALSE.
useASPP	TRUE or FALSE. Whether to replace the sequential bottleneck (EfficientNet-B2 stages 6-8) with an Atrous Spatial Pyramid Pooling (ASPP) module. If TRUE, the ASPP output channel count is set by btnChn. Default is FALSE.
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.
dcChn	Vector of 5 integers defining the number of output feature maps for each of the 5 decoder blocks. Default is 256, 128, 64, 32, and 16.
btnChn	Number of output channels from the ASPP bottleneck when useASPP = TRUE. Ignored when useASPP = FALSE (the sequential bottleneck always outputs 352 channels). Default is 256.
dilRates	Vector of 3 dilation rates for the ASPP module. Default is 6, 12, and 18.
dilChn	Vector of 4 channel counts for each ASPP branch. Default is 256 for each.
negative_slope	If actFunc = "lrelu", specifies the negative slope term. Default is 0.01.
stageLRs	Optional numeric vector of length 11 specifying a base learning rate for each stage: encoder stages e1, e2, e3, e4, e5, the bottleneck, and decoder stages d1, d2, d3, d4, d5 (in that order). Call model$get_param_groups() to obtain optimizer parameter groups with per-stage learning rates. Default is NULL (single learning rate).

Details

Define a UNet architecture with an EfficientNet-B2 backbone or encoder. The architecture has 5 encoder blocks (e1-e5), a bottleneck, and 5 decoder blocks (d1-d5). The final decoder block (d5) uses the original input image as its skip connection, matching the design of `defineMobileUNet`. The user can load ImageNet weights, apply averaged ImageNet weights when the input channel count differs from 3, replace the bottleneck with an ASPP module, add attention gates along the skip connections, freeze the encoder, and enable deep supervision.

Value

EfficientUNetB2 model instance as torch nn_module.

`defineMobileUNet` *defineMobileUNet*

Description

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

Usage

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

Arguments

`inChn`	Number of input channels or predictor variables. 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.
`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. T he default is TRUE. If TRUE, only the decoder component is trained.

`avgImNetWeights`
: TRUE or FALSE. If three predictor variables are provided and ImageNet weights are used, whether or not to use the original weights or average them. Default is FALSE.

`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; "gelu" = GELU. 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.

`stageLRs`
: Optional numeric vector of length 11 specifying a base learning rate for each stage: encoder stages e1, e2, e3, e4, e5, the bottleneck, and decoder stages d1, d2, d3, d4, d5 (in that order). Call `model$get_param_groups()` to obtain optimizer parameter groups with per-stage learning rates. Default is NULL (single learning rate).

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). If ImageNet weights are used and more then three predictor variables are provided, ImageNet weights in the layer of the encoder block are averaged. If three channels or predictor variables are provided, the user can specify to user the ImageNet weights or average them.

Value

ModileUNet model instance as torch nn_module

`defineRegressionLoss` *defineRegressionLoss*

Description

Define a regression loss function for use in a luz training loop.

Usage

```
defineRegressionLoss(type = "mse", delta = 1, reduction = "mean")
```

Arguments

`type`	Character string specifying the loss type. One of `"mse"` (default), `"mae"`, `"huber"`, or `"log_cosh"`.
`delta`	Positive numeric scalar controlling the quadratic-to-linear transition in the Huber loss. Residuals with absolute value below `delta` are penalised quadratically; those above are penalised linearly. Ignored for all other loss types. Default is `1.0`.
`reduction`	Character string controlling how element-wise losses are aggregated across the batch. One of `"mean"` (default), `"sum"`, or `"none"` (returns the full element-wise loss tensor without reduction).

Details

Returns a `torch::nn_module` that computes one of four common regression losses between a prediction tensor and a target tensor. The returned object is passed directly to the `loss` argument of `luz::setup()`.

The four supported loss types are:

`"mse"` Mean Squared Error.

$$\frac{1}{n}\sum_i (\hat{y}_i - y_i)^2$$

Penalises large errors quadratically. Sensitive to outliers because the squared term magnifies large residuals.

`"mae"` Mean Absolute Error.

$$\frac{1}{n}\sum_i |\hat{y}_i - y_i|$$

More robust to outliers than MSE. The gradient is constant in magnitude for all non-zero residuals, which can slow convergence near the optimum.

`"huber"` Huber loss.

$$L_\delta(r) = \begin{cases} \frac{1}{2}r^2 & |r| \le \delta \\ \delta\big(|r| - \frac{1}{2}\delta\big) & |r| > \delta \end{cases}$$

where $r = \hat{y} - y$. Quadratic for small residuals (like MSE) and linear for large residuals (like MAE), combining outlier robustness with smooth gradients near zero. The transition point is set by `delta`.

`"log_cosh"` Log-Cosh loss.

$$\frac{1}{n}\sum_i \log\big(\cosh(\hat{y}_i - y_i)\big)$$

Approximates $\frac{1}{2}r^2$ for small residuals and $|r| - \log 2$ for large residuals. Doubly differentiable everywhere, which benefits second-order optimisers and produces very smooth loss surfaces. Computed in a numerically stable form: $|r| + \log(1 + e^{-2|r|}) - \log 2$.

Value

An instantiated `torch::nn_module` whose `forward` method accepts `(input, target)` tensors of any matching shape and returns the scalar (or element-wise when `reduction = "none"`) loss.

Examples

```
## Not run:
# MSE loss (default)
loss_fn <- defineRegressionLoss()

# MAE loss
loss_fn <- defineRegressionLoss(type = "mae")

# Huber loss with a tighter transition point
loss_fn <- defineRegressionLoss(type = "huber", delta = 0.5)

# Log-Cosh loss
loss_fn <- defineRegressionLoss(type = "log_cosh")

# Use in a luz training loop
fitted <- myModel |>
  luz::setup(
    loss      = defineRegressionLoss(type = "huber", delta = 1.0),
    optimizer = torch::optim_adamw
  ) |>
  luz::fit(data = trainDL, epochs = 30, valid_data = valDL)

## End(Not run)
```

`defineResNetUNet` *defineResNetUNet*

Description

Define a UNet architecture for geospatial semantic segmentation with a ResNet backbone.

Usage

```
defineResNetUNet(
  inChn = 3,
  nCls = 3,
  backbone = "resnet34",
  pretrainedEncoder = TRUE,
  freezeEncoder = FALSE,
  avgImNetWeights = FALSE,
  actFunc = "relu",
  useAttn = FALSE,
  useDS = FALSE,
  dcChn = c(256, 128, 64, 32, 16),
  negative_slope = 0.01,
  stageLRs = NULL
)
```

Arguments

inChn	Number of input channels or predictor variables. 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.
backbone	ResNet variant to use as the encoder: "resnet18", "resnet34", "resnet50", "resnet101", or "resnet152". Default is "resnet34".
pretrainedEncoder	TRUE or FALSE. Whether to initialize the encoder with pre-trained ImageNet weights. Default is TRUE.
freezeEncoder	TRUE or FALSE. Whether to freeze the encoder during training. If TRUE, only the decoder is trained. Default is FALSE.
avgImNetWeights	TRUE or FALSE. If three predictor variables are provided and ImageNet weights are used, whether to average the stem weights rather than use the originals. If the input has a number of channels other than 3, the stem weights are always averaged regardless of this argument. Default is FALSE.
actFunc	Activation function used in the decoder (the ResNet encoder layers are not affected; they use ReLU internally). "relu" = rectified linear unit (ReLU); "lrelu" = leaky ReLU; "swish" = swish; "gelu" = GELU. Default is "relu".
useAttn	TRUE or FALSE. Whether to add attention gates along the skip connections. Default is FALSE.
useDS	TRUE or FALSE. Whether 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. If FALSE, only the final prediction at the original resolution is returned. Default is FALSE.
dcChn	Vector of 5 integers defining the number of output feature maps for each of the 5 decoder blocks. Default is 256, 128, 64, 32, and 16.
negative_slope	If actFunc = "lrelu", specifies the negative slope term to use. Default is 0.01.
stageLRs	Optional numeric vector of length 11 specifying a base learning rate for each stage: encoder stages e1, e2, e3, e4, e5, the bottleneck, and decoder stages d1, d2, d3, d4, d5 (in that order). Call `model$get_param_groups()` to obtain optimizer parameter groups with per-stage learning rates. Default is NULL (single learning rate).

Details

Define a UNet architecture with a torchvision ResNet encoder (ResNet-18, -34, -50, -101, or -152), following the same design as `defineMobileUNet`: 5 encoder stages (the shallowest is an identity pass-through of the input image, so the final decoder block skips from the raw input), a bottleneck, and 5 decoder blocks with transpose-convolution upsampling. Optional attention gates along the skip connections (`useAttn`) and deep supervision (`useDS`) are available.

The encoder is tapped as: stem (`conv1` + `bn1` + `relu`, stride 2), `maxpool` + `layer1` (stride 4), `layer2` (stride 8), `layer3` (stride 16), and `layer4` as the bottleneck (stride 32). Channel counts are taken

from the chosen backbone (64/64/128/256/512 for ResNet-18/34; 64/256/512/1024/2048 for ResNet-50/101/152).

If ImageNet weights are used and the number of input channels differs from 3, or if `avgImNetWeights = TRUE`, the stem convolution weights are averaged over the input-channel dimension and expanded to `inChn` channels.

Value

ResNetUNet model instance as torch nn_module.

`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,
  probRotation = 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),
  useEdgeDist = FALSE
)
```

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.
probRotation	0 to 1. Probability of applying rotation by 0-, 90-, 180- or 270-degrees. 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.

useEdgeDist TRUE or FALSE. Whether to load and return the distance-to-class-boundary grid for each chip. Requires a `distPth` column in `chpDF`, produced by `makeChipsDF` with `useEdgeDist = TRUE`. When TRUE, the returned list includes a `dist` element of shape (1, H, W). Default is FALSE.

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).

When useEdgeDist = TRUE the dataset expects chpDF to contain a distPth column (produced by makeChipsDF with useEdgeDist = TRUE). Each distance chip is loaded as a single-band float32 tensor of shape (1, H, W). Spatial augmentations (flips, rotation) are applied consistently across the image, mask, and distance grid. Photometric augmentations (brightness, contrast, gamma, hue, saturation) are applied to the image only and do not affect the distance grid. When useEdgeDist = TRUE the returned list contains a third element named dist; this should be passed as the third argument to the loss function.

Value

A dataset object that can be provided to torch::dataloader(). Each item is a named list with elements image and mask. When useEdgeDist = TRUE a third element dist is also returned.

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,
  useEdgeDist=TRUE,
  doAugs = TRUE,
  maxAugs = 1,
  probVFlip = .5,
  probHFlip = .5,
  probBrightness = 0,
  probContrast = 0,
  probGamma = 0,
  probHue = 0,
  probRotation = 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)
```

---

defineTerrainSeg *defineTerrainSeg*

Description

CNN-based semantic segmentation wrapper that generates LSPs from a DTM and passes them to a user-supplied trainable segmentation model.

Usage

```
defineTerrainSeg(
  segMod,
  cellSize = 1,
  spatDim = 640,
  tCrop = 64,
  doGP = FALSE,
  innerRadius = 2,
  outerRadius = 10,
  hsRadius = 50,
  smoothRadius = 11
)
```

Arguments

segMod	An already-instantiated trainable segmentation model (torch nn_module). Must accept the number of input channels produced by the LSP module: 6 when doGP = FALSE, 31 when doGP = TRUE.
cellSize	Input resolution of DTM data in map units. Default is 1.
spatDim	Spatial dimension (height = width) of the input chip in cells. Default is 640.
tCrop	Number of rows and columns to crop from each side of the LSP output before passing to the segmentation model. This removes edge artefacts introduced by the convolution-based LSP calculations. Default is 64.
doGP	Whether to compute Gaussian Pyramids of the DTM and derive LSPs at multiple scales. Default is FALSE. If FALSE, 6 LSPs are passed to the model. If TRUE, 31 LSPs are passed.
innerRadius	Inner radius (cells) of the annulus window used for local TPI calculation. Default is 2.
outerRadius	Outer radius (cells) of the annulus window used for local TPI calculation. Default is 10.
hsRadius	Radius (cells) of the circular moving window for hillslope TPI calculation. Default is 50.
smoothRadius	Radius (cells) of the circular moving window used to smooth the DTM before curvature calculations. Default is 11.

Details

Define a CNN-based semantic segmentation model for landform extraction or classification. The module generates land surface parameters (LSPs) from the input DTM, crops the LSP tensor to remove edge artefacts, and then passes the result to an externally instantiated trainable segmentation model. The model assumes a single-band DTM of elevation measurements as input.

When doGP = FALSE the LSP module produces 6 channels; when doGP = TRUE it produces 31 channels. The segmentation model supplied via segMod must be instantiated with a matching inChn value and with an inChn that accounts for the spatial reduction caused by tCrop (input chip size minus 2 * tCrop).

Value

A `terrainSeg` `nn_module` wrapping the LSP pipeline and the supplied segmentation model.

`defineUNet` *defineUNet*

Description

Define a UNet architecture for geospatial semantic segmentation.

Usage

```
defineUNet(
  inChn = 3,
  nCls = 3,
  actFunc = "relu",
  encBlocks = "double",
  btnBlock = "double",
  decBlocks = "double",
  useAttn = FALSE,
  useSE = FALSE,
  useCBAM = FALSE,
  useRes = FALSE,
  useASPP = FALSE,
  useDS = FALSE,
  enChn = c(16, 32, 64, 128),
  dcChn = c(128, 64, 32, 16),
  btnChn = 256,
  dilRates = c(6, 12, 18),
  dilChn = c(256, 256, 256, 256),
  negative_slope = 0.01,
  seRatio = 8,
  cbamRatio = 8,
  cbamKernelSize = 7,
  expRatio = 4,
  stageLRs = NULL
)
```

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 for the non-ConvNeXt blocks. "relu" = rectified linear unit (ReLU); "lrelu" = leaky ReLU; "swish" = swish; "gelu" = GELU. Default is "relu".
encBlocks	Block implementation for the four encoder blocks. Either a single value applied to all four blocks or a length-4 character vector, with entries "double", "dws", or "convnext". Default is "double".
btnBlock	Block implementation for the bottleneck: "double", "dws", "convnext", or "aspp". Default is "double".
decBlocks	Block implementation for the four decoder blocks. Either a single value applied to all four blocks or a length-4 character vector, with entries "double", "dws", or "convnext". Default is "double".
useAttn	TRUE or FALSE. Whether to add attention gates along the skip connections. Default is FALSE or no attention gates are added. Ignored (forced FALSE) when useCBAM = TRUE.
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. Ignored (forced FALSE) when useCBAM = TRUE.
useCBAM	TRUE or FALSE. Whether to apply a Convolutional Block Attention Module (CBAM) to each encoder skip connection before it is concatenated with the upsampled decoder features. When TRUE, useSE and useAttn are forced to FALSE. Default is FALSE.
useRes	TRUE or FALSE. Whether to include residual connections in the "double" and "dws" encoder, decoder, and bottleneck blocks and in the ASPP bottleneck. Has no effect on "convnext" blocks (always residual). Default is FALSE or no residual connections are included.
useASPP	Deprecated. Use btnBlock = "aspp" instead. If TRUE and btnBlock is left at its default, the bottleneck is set to "aspp". Default is FALSE.
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 3 values specifying the dilation rates used in the ASPP module. Default is 6, 12, and 18.
dilChn	Vector of 4 values specifying the number of channels to produce at each dilation rate within the ASPP module. Default is 256 for each dilation rate.
negative_slope	If actFunc = "lrelu", specifies the negative slope term to use. Default is 0.01.
seRatio	Ratio to use in the squeeze and excitation modules. The default is 8.

`cbamRatio` Channel reduction ratio used in the CBAM channel attention MLP. Default is 8.

`cbamKernelSize` Kernel size for the CBAM spatial attention convolution. Must be odd. Default is 7.

`expRatio` Inverted-bottleneck expansion ratio for "convnext" blocks. Default is 4.

`stageLRs` Optional numeric vector of length 9 specifying a base learning rate for each stage of the network: encoder stages e1, e2, e3, e4 (each paired with its squeeze-and-excitation or CBAM module when enabled), the bottleneck, and decoder stages d1, d2, d3, d4 (in that order). When provided, call `model$get_param_groups()` to obtain a list of optimizer parameter groups with per-stage learning rates. Default is NULL (single learning rate).

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.

Each encoder block, each decoder block, and the bottleneck can independently use one of several block implementations via `encBlocks`, `decBlocks`, and `btnBlock`:

- `"double"` — the standard double 3x3 convolution block (default everywhere). Becomes a residual double convolution block when `useRes = TRUE`.
- `"dws"` — a double convolution block whose second 3x3 convolution is replaced with a depthwise separable convolution (`dwsDoubleConv`). Becomes a residual variant (`dwsDoubleConvR`) when `useRes = TRUE`.
- `"convnext"` — a ConvNeXt-style block (depthwise 7x7, LayerNorm, 1x1 inverted-bottleneck expansion, GELU, 1x1 projection, residual; `doubleConvBlkCNX`). Always residual and always GELU, so `useRes` and `actFunc` do not affect these blocks.
- `"aspp"` — (`btnBlock` only) a modified atrous spatial pyramid pooling module for multi-scale context.

Users can also choose to (1) replace all ReLU activation functions with leaky ReLU, swish, or GELU in the non-ConvNeXt blocks, (2) implement attention gates along the skip connections (`useAttn`), (3) implement squeeze-and-excitation modules within the encoder (`useSE`), (4) apply Convolutional Block Attention Modules (CBAM) to the skip connections (`useCBAM`), (5) add residual connections within the "double" blocks and ASPP bottleneck (`useRes`), and/or (6) implement deep supervision using predictions generated at each stage in the decoder (`useDS`). Because CBAM already performs both channel and spatial recalibration, setting `useCBAM = TRUE` forces `useSE` and `useAttn` to FALSE.

Value

Unet model instance as torch nn_module

defineUNet3p *defineUNet3p*

Description

Define a UNet3+ architecture for use in a luz training loop.

Usage

```
defineUNet3p(
  inChn = 3,
  nCls = 2,
  actFunc = "lrelu",
  encBlocks = "double",
  btnBlock = "double",
  skipProj = "conv3x3",
  useRes = FALSE,
  enChn = c(16, 32, 64, 128),
  outChn = 64,
  btnChn = 256,
  useASPP = FALSE,
  dilRates = c(6, 12, 18),
  dilChn = c(256, 256, 256, 256),
  negative_slope = 0.01,
  expRatio = 4,
  useDS = FALSE,
  stageLRs = NULL
)
```

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	Activation function to use throughout the non-ConvNeXt blocks. "relu" = ReLU; "lrelu" = leaky ReLU; "swish" = SiLU/Swish; "gelu" = GELU. Default is "lrelu".
encBlocks	Block implementation for the four encoder blocks. Either a single value applied to all four blocks or a length-4 character vector, with entries "double", "dws", or "convnext". Default is "double".
btnBlock	Block implementation for the bottleneck: "double", "dws", "convnext", or "aspp". Default is "double".
skipProj	Channel-unification convolution used on each branch of the full-scale skip connections: "conv3x3" (original UNet3+) or "conv1x1" (lighter). Default is "conv3x3".

useRes	TRUE or FALSE. Whether to include residual connections in the "double" and "dws" encoder blocks, the "double"/"dws"/"aspp" bottleneck, and the four per-node decoder feature-aggregation blocks (d1Blk-d4Blk). Has no effect on "convnext" blocks (always residual). Default is FALSE.
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.
outChn	Number of channels each skip-connection branch is projected to before concatenation. Each decoder node therefore operates on 5 * outChn channels. Default is 64.
btnChn	Number of output feature maps from the bottleneck block. Default is 256.
useASPP	Deprecated. Use btnBlock = "aspp" instead. If TRUE and btnBlock is left at its default, the bottleneck is set to "aspp". Default is FALSE.
dilRates	Vector of 3 values specifying the dilation rates used in the ASPP module. Default is 6, 12, and 18.
dilChn	Vector of 4 values specifying the number of channels to produce at each dilation rate within the ASPP module. Default is 256 for each dilation rate.
negative_slope	Specifies the negative slope term for leaky ReLU activation. Default is 0.01.
expRatio	Inverted-bottleneck expansion ratio for "convnext" blocks. Default is 4.
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.
stageLRs	Optional numeric vector of length 9 specifying a base learning rate for each stage: encoder stages encoder1, encoder2, encoder3, encoder4 (including their cross-scale projections), the bottleneck (including its decoder projections), and decoder stages d1, d2, d3, d4 (in that order). Call model$get_param_groups() to obtain optimizer parameter groups. Default is NULL (single learning rate).

Details

Define a UNet3+-like architecture with full-scale skip connections. The user can specify the number of output feature maps from each encoder block, the per-branch channel count used in the full-scale skip connections (outChn), and the bottleneck. Deep supervision can also be implemented. A variable number of input predictor variables and output classes can be defined.

As in the original UNet3+, downsampling within the full-scale skip connections uses non-overlapping max pooling and upsampling uses bilinear interpolation (there are no transpose convolutions). Each resampled branch is passed through a channel-unification convolution before the branches are concatenated; skipProj selects a 3x3 convolution ("conv3x3", the original UNet3+ design) or a lighter 1x1 convolution ("conv1x1"). The per-node feature aggregation block that follows each concatenation is always a 3x3 convolution.

Each of the four encoder blocks and the bottleneck can independently use one of three block implementations via encBlocks and btnBlock, matching defineUNet:

- "double" - the standard double 3x3 convolution block (default). Becomes a residual double convolution block when useRes = TRUE.

- "dws" - a double convolution block whose second 3x3 convolution is replaced with a depth-wise separable convolution (dwsDoubleConv). Becomes a residual variant (dwsDoubleConvR) when useRes = TRUE.
- "convnext" - a ConvNeXt-style block (doubleConvBlkCNX); always residual and always GELU, so useRes and actFunc do not affect these blocks.

btnBlock additionally accepts "aspp" for an atrous spatial pyramid pooling bottleneck, which also respects useRes (asppBlk vs. asppBlkR). The four per-node feature-aggregation blocks (d1Blk through d4Blk) likewise become residual when useRes = TRUE; since their input and output channel counts are always equal, the residual skip there is a parameter-free identity.

The architecture was inspired by:

Huang, H., Lin, L., Tong, R., Hu, H., Zhang, Q., Iwamoto, Y., Han, X., Chen, Y.W. and Wu, J., 2020, May. Unet 3+: A full-scale connected unet for medical image segmentation. In ICASSP 2020-2020 IEEE international conference on acoustics, speech and signal processing (ICASSP) (pp. 1055-1059). IEEE.

Value

UNet3+ model using nn_module().

---

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,
  cropFactorMsk = 0,
  cropFactorPred = 0,
  lambda = 0.5,
  gamma = 0.5,
  delta = 0.6,
  smooth = 1e-08,
  zeroStart = TRUE,
  clsWghtsDist = 1,
  clsWghtsReg = 1,
  useLogCosH = FALSE,
  useEdgeDist = FALSE,
  prioritizeEdge = TRUE,
  edgeMode = "threshold",
```

```
  distThreshold = 10,
  minWght = 0.5,
  maxWght = 1,
  applyDistWght = "both",
  labelSmoothing = 0,
  classAvgDist = FALSE,
  device = "cuda"
)
```

Arguments

nCls	Number of classes being differentiated.
cropFactorMsk	Number of rows and columns of cells to not include for mask in assessment to minimize edge effects. Default is 0 or no cropping.
cropFactorPred	Number of rows and columns of cells to not include for prediction in assessment to minimize edge effects. Default is 0 or no cropping.
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.
useEdgeDist	TRUE or FALSE. Whether to apply distance-to-class-boundary weights during loss computation. When TRUE a `dist` tensor must be supplied to `forward`. Default is FALSE.

prioritizeEdge	TRUE or FALSE. When useEdgeDist = TRUE, controls which pixels receive the higher weight. If TRUE (default), pixels near class boundaries (small distance) receive maxWght and pixels far from boundaries receive minWght. If FALSE, the weighting is reversed so that non-edge pixels are emphasised.
edgeMode	Character string specifying how the distance is converted to a per-pixel weight. One of "threshold", "linear", or "gaussian". Default is "threshold". See the Details section for a full description of each mode.
distThreshold	Numeric. The distance value (in map units or pixel units, matching the distance raster) that defines the boundary of the high-weight zone. Default is 10.
minWght	Numeric. The minimum per-pixel weight applied to loss contributions. Must be less than maxWght. Default is 0.5.
maxWght	Numeric. The maximum per-pixel weight applied to loss contributions. Must be greater than minWght. Default is 1.0.
applyDistWght	Character string controlling which loss component the edge weights are applied to. One of "distribution" (distribution-based/CE loss only), "region" (region-based/Tversky loss only), or "both" (default).
labelSmoothing	Numeric value in [0, 1). When greater than 0, this amount of probability mass is spread uniformly across all classes when computing the cross-entropy, preventing the model from becoming overconfident. The focal modifier is always derived from the true-class softmax probability gathered directly from the network output, so focal weighting is not distorted by the smoothed targets. Default is 0 (no smoothing).
classAvgDist	TRUE or FALSE. When FALSE (default), the distribution loss is computed as a pixel-level weighted mean of the focal cross-entropy, with clsWghtsDist rescaling each pixel's contribution by its true class weight. When TRUE, the focal cross-entropy is first averaged within each class and the resulting per-class means are then combined as a weighted average using clsWghtsDist. The per-class approach mirrors the macro-averaging philosophy already used by the region-based loss and can give better gradient signal to under-represented classes under severe class imbalance. Note that if a class has no pixels in a given batch its contribution is effectively zero for that step, consistent with the behaviour of the region-based loss.
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, (3) including an option to apply a logcosh transform to the region-based loss, (4) optionally incorporating distance-to-class-boundary weights computed by makeBoundaryDistance, (5) an option to compute the distribution loss as a weighted average of

per-class means rather than as a single pixel-level weighted mean, and (6) optional label smoothing on the distribution 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.

When `labelSmoothing > 0` the cross-entropy is computed against soft targets that spread `labelSmoothing` probability mass uniformly across all classes, reducing the model's incentive to become overconfident. The focal modifier is kept anchored to the true-class softmax probability gathered directly from the network output, so the focal weighting continues to reflect genuine model confidence rather than being distorted by the smoothed targets. Label smoothing is applied only to the distribution loss; the region-based (Tversky) loss is not affected.

When `useEdgeDist = TRUE` a distance-to-class-boundary tensor must be passed as the third argument to `forward(pred, target, dist)`. The tensor should have shape (N, 1, H, W) and contain the per-pixel distances produced by `makeBoundaryDistance`, stacked into a batch. Pixels at a class boundary have distance 0; distance increases away from boundaries.

Three edge-weighting modes are available via `edgeMode`:

- `"threshold"`: pixels within `distThreshold` receive one weight; pixels beyond receive the other. When `prioritizeEdge = TRUE` the inside weight is `maxWght` and the outside weight is `minWght`; the relationship is reversed when `prioritizeEdge = FALSE`.
- `"linear"`: weight changes linearly from `maxWght` at distance 0 to `minWght` at `distThreshold` (when `prioritizeEdge = TRUE`), or from `minWght` to `maxWght` (when `prioritizeEdge = FALSE`). Beyond `distThreshold` the weight is clamped to the far-end value.
- `"gaussian"`: weight follows a Gaussian curve centred at distance 0. The curve uses sigma = distThreshold / 3 so that the weight is approximately `minWght` at `distThreshold` and falls to exactly `minWght` beyond it. Direction is controlled by `prioritizeEdge` as above.

Value

Loss metric for use in training process.

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,
  cropFactorMsk = 0,
  cropFactorPred = 0,
  dsWghts = c(0.6, 0.2, 0.1, 0.1),
  weight_env = NULL,
  lambda = 0.5,
  gamma = 0.5,
  delta = 0.6,
  smooth = 1e-08,
  zeroStart = TRUE,
  clsWghtsDist = 1,
  clsWghtsReg = 1,
  useLogCosH = FALSE,
  useEdgeDist = FALSE,
  prioritizeEdge = TRUE,
  edgeMode = "threshold",
  distThreshold = 10,
  minWght = 0.5,
  maxWght = 1,
  applyDistWght = "both",
  labelSmoothing = 0,
  classAvgDist = FALSE,
  device = "cuda"
)
```

Arguments

nCls	Number of classes being differentiated.
cropFactorMsk	Number of rows and columns of cells to not include for mask in assessment to minimize edge effects. Default is 0 or no cropping.
cropFactorPred	Number of rows and columns of cells to not include for prediction in assessment to minimize edge effects. Default is 0 or no cropping.
dsWghts	Vector of 4 weights applied to the losses at each spatial resolution. Used only when weight_env is NULL. The default is c(.6, .2, .1, .1) where larger weights are placed on the results at a higher spatial resolution.

weight_env	Optional environment created by make_ds_weights(). When supplied, the loss reads weight_env$values on every forward pass so that callback_ds_weight_decay() can decay the auxiliary weights during training without rebuilding the loss. Takes precedence over dsWghts. Default is NULL.
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.
useEdgeDist	TRUE or FALSE. Whether to apply distance-to-class-boundary weights during loss computation. When TRUE a dist tensor must be supplied to forward. Default is FALSE.
prioritizeEdge	TRUE or FALSE. When useEdgeDist = TRUE, controls which pixels receive the higher weight. If TRUE (default), pixels near class boundaries receive maxWght and pixels far from boundaries receive minWght. If FALSE, the weighting is reversed.
edgeMode	Character string specifying how the distance is converted to a per-pixel weight. One of "threshold", "linear", or "gaussian". Default is "threshold".
distThreshold	Numeric. The distance value that defines the boundary of the high-weight zone. Default is 10.
minWght	Numeric. The minimum per-pixel weight. Must be less than maxWght. Default is 0.5.
maxWght	Numeric. The maximum per-pixel weight. Must be greater than minWght. Default is 1.0.

applyDistWght	Character string controlling which loss component the edge weights are applied to. One of "distribution", "region", or "both" (default).
labelSmoothing	Numeric value in [0, 1). Amount of label smoothing applied to the cross-entropy component of the distribution loss. The focal modifier is derived from the true-class softmax probability and is not affected by smoothing. Default is 0 (no smoothing). See defineUnifiedFocalLoss for full details.
classAvgDist	TRUE or FALSE. When TRUE, the distribution loss is computed as a weighted average of per-class focal cross-entropy means rather than a pixel-level weighted mean. See defineUnifiedFocalLoss for a full description. 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, (3) including an option to apply a logcosh transform to the region-based loss, (4) optionally incorporating distance-to-class-boundary weights computed by makeBoundaryDistance, (5) an option to compute the distribution loss as a weighted average of per-class means rather than as a single pixel-level weighted mean, and (6) optional label smoothing on the distribution 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.

When useEdgeDist = TRUE a distance-to-class-boundary tensor must be passed as the third argument to forward(pred, target, dist). The same distance tensor is applied to all decoder-output losses. See defineUnifiedFocalLoss for a full description of the edge-distance weighting options.

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,
  cropFactorMsk = 0,
  cropFactorPred = 0,
  smooth = 1,
  mode = "multiclass",
  biThresh = 0.5,
  clsWghts = rep(1, nCls),
```

```
  zeroStart = TRUE,
  usedDS = FALSE
)
```

Arguments

nCls	Number of classes being differentiated.
cropFactorMsk	Number of rows and columns of cells to not include for mask in assessment to minimize edge effects. Default is 0 or no cropping.
cropFactorPred	Number of rows and columns of cells to not include for prediction in assessment to minimize edge effects. Default is 0 or no cropping.
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

```
if (torch::torch_is_installed()) {
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,
  cropFactorMsk = 0,
  cropFactorPred = 0,
  smooth = 1,
  mode = "multiclass",
  biThresh = 0.5,
  zeroStart = TRUE,
  usedDS = FALSE
)
```

Arguments

nCls	Number of classes being differentiated.
cropFactorMsk	Number of rows and columns of cells to not include for mask in assessment to minimize edge effects. Default is 0 or no cropping.
cropFactorPred	Number of rows and columns of cells to not include for prediction in assessment to minimize edge effects. Default is 0 or no cropping.
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

```
if (torch::torch_is_installed()) {
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,
  cropFactorMsk = 0,
  cropFactorPred = 0,
  smooth = 1,
  mode = "multiclass",
  biThresh = 0.5,
  zeroStart = TRUE,
  clsWghts = rep(1, nCls),
  usedDS = FALSE
)
```

Arguments

`nCls`	Number of classes being differentiated.
`cropFactorMsk`	Number of rows and columns of cells to not include for mask in assessment to minimize edge effects. Default is 0 or no cropping.
`cropFactorPred`	Number of rows and columns of cells to not include for prediction in assessment to minimize edge effects. Default is 0 or no cropping.

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

```
if (torch::torch_is_installed()) {
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,
  cropFactorMsk = 0,
  cropFactorPred = 0,
  smooth = 1,
  mode = "multiclass",
  biThresh = 0.5,
  zeroStart = TRUE,
  clsWghts = rep(1, nCls),
  usedDS = FALSE
)
```

Arguments

nCls	Number of classes being differentiated.
cropFactorMsk	Number of rows and columns of cells to not include for mask in assessment to minimize edge effects. Default is 0 or no cropping.
cropFactorPred	Number of rows and columns of cells to not include for prediction in assessment to minimize edge effects. Default is 0 or no cropping.
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

```
if (torch::torch_is_installed()) {
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()
}
```

---

makeAspect *makeAspect*

Description

Calculate aspect or slope orientation or a related metric from a digital terrain model (DTM) using torch

Usage

```
makeAspect(
  dtm,
  cellSize = 1,
  flatThreshold = 1,
  mode = "aspect",
  writeRaster = FALSE,
  outName,
  device = "cpu"
)
```

Arguments

dtm	Input SpatRaster object representing bare earth surface elevations.
cellSize	Resolution of raster grid. Default is 1 m.
flatThreshold	Maximum slope to be re-coded to flat and assigned an aspect value of -1. Default is 1-degree.
mode	"aspect", "northness", "eastness", "trasp", or "sei". Default is "aspect".
writeRaster	TRUE or FALSE. Save output to disk. Default is TRUE.
outName	Name of output raster with full file path and extension.
device	"cpu" or "cuda". Use "cuda" for GPU computation. Without using the GPU, implementation will not be significantly faster than using non-tensor-based computation. Defaults is "cpu".

Details

Calculate topographic aspect or slope orientation or a related metric using torch in degree units. Processing on the GPU can be much faster than using base R and non-tensor-based calculations. "aspect" = topographic aspect in degree units where flat slopes are coded to -1; "northness" = cosine of aspect in radians; "eastness" = sine of aspect in radians; "trasp" = topographic radiation aspect index; "sei" = site exposure index.

Value

Single-band SpatRaster object. If writeRaster = TRUE, the raster is also written to disk.

Examples

```
## Not run:
pth <- "OUTPUT PATH"
dtm <- rast(paste0(pth, "dtm.tif"))
aspR <- makeAspect(dtm, cellSize=1,
flatThreshold=1,
mode= "aspect",
writeRaster=TRUE,
outName=paste0(pth, "asp.tif"),
device="cuda")

## End(Not run)
```

---

makeBoundaryDistance *makeBoundaryDistance*

Description

Generate distance-to-class-boundary grids from mask chips

Usage

```
makeBoundaryDistance(chipsDir, noDataValue = 0)
```

Arguments

chipsDir	Full or relative path to the chips directory that contains a "masks" subfolder. This is the same directory that was used as the `outDir` argument to `makeChips`. Do not include a trailing slash.
noDataValue	Numeric scalar. The value written to every pixel of a distance grid when the corresponding mask chip contains no class boundaries (i.e. the chip belongs to a single class throughout). Default is `0`.

Details

This function reads raster mask chips from the "masks" subfolder of a chips directory created by `makeChips` and generates a corresponding distance surface for each mask. Each output raster encodes, for every pixel, the distance to the nearest class boundary. A class boundary is any pixel that is adjacent (in an 8-connected neighborhood) to at least one pixel belonging to a different class. Boundary pixels themselves receive a value of 0; all other pixels receive the Euclidean distance (in pixel units, unless the raster has a projected coordinate system, in which case map units are used) to the nearest boundary pixel. Output grids are written to a "dist" subfolder created inside the provided chips directory. Output files are named identically to their corresponding masks and are saved in TIFF format.

The function automatically detects whether the chips directory uses a flat layout (`masks/`) produced by `makeChips` with `mode = "All"` or `mode = "Positive"`, or a divided layout (`masks/positive/` and `masks/background/`) produced with `mode = "Divided"`. For the divided layout, distance grids are written to `dist/positive/` and `dist/background/` respectively.

Value

Distance-to-boundary raster files written to disk in TIFF format inside a "dist" subfolder (flat layout) or "dist/positive/" and "dist/background/" subfolders (divided layout). No R object is returned.

Examples

```
## Not run:
makeBoundaryDistance(chipsDir = "C:/data/chips")

# Fill single-class chips with a large sentinel instead of 0
makeBoundaryDistance(chipsDir = "C:/data/chips", noDataValue = 999)

## End(Not run)
```

`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". For multiclass problems, "All" or "Positive" are recommended.
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 generate 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 supports both binary and multiclass classification problems. For binary classification the background class should be coded 0 and the positive class coded 1. For multiclass problems, class codes should be sequential integers (e.g., 0 to n-1 or 1 to n). In all modes, chips or masks that contain any NA or NoData cells are excluded.

Three modes are available. If "All" is used, all image chips are generated, including those that contain only background pixels (class code 0). 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 contain at least one non-background pixel (i.e., any pixel with a class code greater than 0) will be produced. This mode is recommended for both binary and multiclass problems where background-only chips should be excluded. 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" mode is used, separate "positive" and "background" folders will be created with "images" and "masks" subfolders. Any chip containing at least one non-background pixel will be written to the "positive" folder; chips containing only background pixels will be written to the "background" folder. This mode is primarily intended for binary classification problems.

Value

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

Examples

```
## Not run:
# Binary classification
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 DIRECTORY IN WHICH TO SAVE CHIPS",
          mode = "Positive",
          useExistingDir = FALSE)

# Multiclass classification
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 DIRECTORY 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,
  useEdgeDist = 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.
useEdgeDist	TRUE or FALSE. Whether to include a `distPth` column pointing to the distance-to-class-boundary grids produced by `makeBoundaryDistance`. Supported for all modes. 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.

If `useEdgeDist = TRUE`, a fourth column `distPth` is added that points to the corresponding distance-to-class-boundary grid produced by `makeBoundaryDistance`. All three modes are supported. For `mode = "All"` or `mode = "Positive"` the paths resolve to `dist/<name>.tif`; for `mode = "Divided"` they resolve to `dist/positive/<name>.tif` or `dist/background/<name>.tif` to match the layout written by `makeBoundaryDistance` when it detects the divided folder structure.

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. If `useEdgeDist = TRUE`, an additional `distPth` column is included.

Examples

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

## End(Not run)
```

`makeChipsMultiClass` *makeChipsMultiClass*

Description

Deprecated. Use `makeChips` with `mode = "Positive"` instead.

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.
`mask`	SpatRaster object or path to single-band mask.
`n_channels`	Number of channels in the input image. Default is 3.
`size`	Size of image chips in 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 output directory. You must include the final forward slash (e.g., "C:/data/chips/").
useExistingDir	TRUE or FALSE. Write into an existing directory. Default is FALSE.

Details

This function is a deprecated wrapper around makeChips. It will be removed in a future version. Replace calls of the form:

```
makeChipsMultiClass(image, mask, ...)
```

with:

```
makeChips(image, mask, ..., mode = "Positive")
```

Value

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

Examples

```
## Not run:
# Deprecated — use makeChips() instead:
makeChips(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",
          mode = "Positive",
          useExistingDir = FALSE)

## End(Not run)
```

---

makeCrv *makeCrv*

Description

Calculate surface curvatures from input digital terrain model (DTM) using torch

Usage

```
makeCrv(
  dtm,
  cellSize = 1,
  mode = "mean",
  smoothRadius = 7,
  writeRaster = FALSE,
  outName,
```

```
  device = "cpu"
)
```

Arguments

dtm	Input SpatRaster object representing bare earth surface elevations.
cellSize	Resolution of raster grid. Default is 1 m.
mode	"mean", "profile", or "planform". Default is "mean".
smoothRadius	radius of circular moving window. Default is 7 cells.
writeRaster	TRUE or FALSE. Save output to disk. Default is TRUE.
outName	Name of output raster with full file path and extension.
device	"cpu" or "cuda". Use "cuda" for GPU computation. Without using the GPU, implementation will not be significantly faster than using non-tensor-based computation. Defaults is "cpu".

Details

Calculate surface curvatures from input digital terrain model (DTM) using torch. "mean" = mean curvature or average of profile and planform curvature; "profile" = curvature in the direction of maximum slope; "planform" = curvature in the direction perpendicular to the direction of maximum slope. Radii are specified using cell counts as opposed to map distance. Gaussian smoothing using a circular window of a user-defined radius is applied prior to calculating curvatures to minimize the impact of local noise/variability.

Value

Single-band SpatRaster object. If writeRaster = TRUE, the raster is also written to disk.

Examples

```
## Not run:
pth <- "OUTPUT PATH"
dtm <- rast(paste0(pth, "dtm.tif"))
crvPro7 <- makeCrv(dtm,
cellSize=1,
mode="profile",
smoothRadius=7,
writeRaster=TRUE,
outName=paste0(pth, "crvPro7.tif"),
device="cuda")

## End(Not run)
```

makeDynamicChipsSF *makeDynamicChipsSF*

Description

Generate a vector point sf object of chip center locations for use with defineDynamicSegDataSet().

Usage

```
makeDynamicChipsSF(
  pntPth,
  pntName,
  featPth,
  featName,
  extentPth,
  extentName,
  extentCrop = 50,
  imgPth,
  imgName,
  doBackground = FALSE,
  backgroundCnt,
  backgroundDist,
  useSeed = FALSE,
  seed,
  doShuffle = FALSE
)
```

Arguments

pntPth	Full folder path or relative path to vector point layer defining chip centers. Must include final forward slash.
pntName	File name of point layer including file extension. Must contain "code" (numeric class code) and "class" (text class name) columns.
featPth	Full folder path or relative path to vector polygon features used as the mask source. Must include final forward slash.
featName	File name of polygon feature layer including file extension.
extentPth	Full folder path or relative path to vector extent. Must include final forward slash.
extentName	File name of extent including file extension.
extentCrop	Amount to crop extent to avoid edge effects. We recommend cropping the extent more than the size of the chips you will use to avoid chips with NA cells. Default is 50 cells.
imgPth	Path to image or predictor variable raster data. Must include the final forward slash.
imgName	File name of image or predictor variable raster data including file extension.

doBackground	Whether or not to randomly select background cells. Default is FALSE.
backgroundCnt	Number of background cells to select.
backgroundDist	Minimum allowed distance between randomly selected background locations and boundaries of reference features.
useSeed	Whether or not to set a random seed for replicability. Default is FALSE.
seed	Random seed value.
doShuffle	Whether or not to shuffle the rows in the resulting table. Default is FALSE.

Details

Requires an input vector point object defining chip center locations, an input vector polygon object of reference features used to generate segmentation masks, a boundary extent, and the path and name of the associated raster predictor variables. Note that the raster predictor variable must cover the full spatial extent defined by the extent object. If not, a smaller extent should be defined for which predictor variables are available.

The input point layer must contain a "code" column of numeric class codes and a "class" column of text class names. The input polygon feature layer is used only as the mask source during training and is not required to share attributes with the point layer, but must also contain "code" and "class" columns if class-specific masking is expected downstream.

The extent can be cropped to avoid generating chips without a full set of cells available. We recommend cropping by a factor larger than 50% the width/height of the desired chip size.

The tool returns a sf dataframe with class name ("class"), class numeric code ("code"), path to the image or predictor variables ("imgPth"), name of the image or predictor variables ("imgName"), the path to the input point layer ("pntPth"), the file name of the input point layer ("pntName"), the path to the input polygon features ("featPth"), the file name of the input polygon features ("featName"), the file path to the extent ("extentPth"), the file name for the extent ("extentName"), and the point geometry ("geometry").

Value

sf dataframe object of center locations for chip creation with associated information as attributes.

---

makeHillshade *makeAspect*

Description

Calculate a hillshde from a digital terrain model (DTM) using torch

Usage

```
makeHillshade(
  dtm,
  cellSize = 1,
  sunAzimuth = 315,
  sunAltitude = 45,
  doMD = FALSE,
  writeRaster = FALSE,
  outName,
  device = "cpu"
)
```

Arguments

dtm	Input SpatRaster object representing bare earth surface elevations.
cellSize	Resolution of raster grid. Default is 1 m.
sunAzimuth	Direction of illuminating source as a compass direction. Default is 315-degrees or northwest.
sunAltitude	Angle of illuminating source above the horizon from 0-degrees (horizon) to 90-degrees (zenith). Default is 45-degrees
doMD	TRUE or FALSE. Whether or not to generate a multidirectional hillshade. Default is FALSE.
writeRaster	TRUE or FALSE. Save output to disk. Default is TRUE.
outName	Name of output raster with full file path and extension.
device	"cpu" or "cuda". Use "cuda" for GPU computation. Without using the GPU, implementation will not be significantly faster than using non-tensor-based computation. Defaults is "cpu".

Details

Calculate a hillshade from a digital terrain model (DTM) using torch. User can specify a illuminating position using an aspect and altitude. A multidirectiontal hillshade is calculated by averaging hillshades with different sun positions ((north X 2Xnorthwest X west X southeast)/5.

Value

Single-band SpatRaster object. If writeRaster = TRUE, the raster is also written to disk.

Examples

```
## Not run:
pth <- "OUTPUT PATH"
dtm <- rast(paste0(pth, "dtm.tif"))
hsR <- makeHillshade(dtm,
cellSize=1,
sunAzimuth=315,
sunAltitude=45,
```

```
doMD = FALSE,
writeRaster=TRUE,
outName=paste0(pth, "hs.tif"), device="cuda")

## 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)
```

makeSlope *makeSlope*

Description

Calculate slope from a digital terrain model (DTM) using torch

Usage

```
makeSlope(dtm, cellSize = 1, writeRaster = FALSE, outName, device = "cpu")
```

Arguments

dtm	Input SpatRaster object representing bare earth surface elevations.
cellSize	Resolution of raster grid. Default is 1 m.
writeRaster	TRUE or FALSE. Save output to disk. Default is TRUE.
outName	Name of output raster with full file path and extension.
device	"cpu" or "cuda". Use "cuda" for GPU computation. Without using the GPU, implementation will not be significantly faster than using non-tensor-based computation. Defaults is "cpu".

Details

Calculate topographic slope using torch in degree units. Processing on the GPU can be much faster than using base R and non-tensor-based calculations.

Value

Single-band SpatRaster object. If writeRaster = TRUE, the raster is also written to disk.

Examples

```
## Not run:
pth <- "OUTPUT PATH"
dtm <- rast(paste0(pth, "dtm.tif"))
slpR <- makeSlope(dtm, cellSize=1, writeRaster=TRUE, outName=paste0(pth, "slp.tif"), device="cuda")

## End(Not run)
```

makeTerrainVisTerra *makeTerrainVisTerra*

Description

Make three band terrain stack from input digital terrain model using terra

Usage

```
makeTerrainVisTerra(
  dtm,
  cellSize = 1,
  innerRadius = 2,
  outerRadius = 10,
  hsRadius = 50,
  writeRaster = FALSE,
  outName
)
```

Arguments

dtm	Input SpatRaster object representing bare earth surface elevations.
cellSize	Resolution of the grid relative to coordinate reference system units (e.g., meters).
innerRadius	inner radius of annulus moving window used for local TPI calculation. Default is 2.
outerRadius	outer radius of annulus moving window used for local TPI calculation.
hsRadius	outer radisu for circular moving window used for hillslope TPI calculation.
writeRaster	TRUE or FALSE. Save output to disk. Default is TRUE.
outName	Name of output raster with full file path and extension.

Details

This function creates a three-band raster stack from an input digital terrain model (DTM) of bare earth surface elevations using terra. This implementation is slower than the torch-based implementation, especially when the torh-based implementation uses GPU-based computation.

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 ange 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:
pth <- "OUTPUT PATH"
dtm <- rast(paste0(pth, "dtm.tif"))
tVisT <- makeTerrainVisTerra(dtm,
cellSize=1,
innerRadius=2,
outerRadius=5,
hsRadius=50,
writeRaster=TRUE,
outName=paste0(pth, "tVisTerra.tif"))

## End(Not run)
```

`makeTerrainVisTorch` *makeTerrainVisTerra*

Description

Make three band terrain stack from input digital terrain model using torch

Usage

```
makeTerrainVisTorch(
  dtm,
  cellSize = 1,
  innerRadius = 2,
  outerRadius = 10,
  hsRadius = 50,
  writeRaster = FALSE,
  outName,
  device = "cpu"
)
```

Arguments

`dtm`	Input SpatRaster object representing bare earth surface elevations.
`cellSize`	Resolution of the grid relative to coordinate reference system units (e.g., meters).
`innerRadius`	inner radius of annulus moving window used for local TPI calculation. Default is 2.
`outerRadius`	outer radius of annulus moving window used for local TPI calculation.
`hsRadius`	outer radisu for circular moving window used for hillslope TPI calculation.

writeRaster	TRUE or FALSE. Save output to disk. Default is TRUE.
outName	Name of output raster with full file path and extension.
device	Device on which to perform calculations. "cpu" or "cuda". Default is "cpu". Recommend "cuda". Recommend "cuda" to speed up calculations.

Details

This function creates a three-band raster stack from an input digital terrain model (DTM) of bare earth surface elevations using torch. This implementation is faster than the terra-based implementation, especially when using GPU-based computation.

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 ange 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:
pth <- "OUTPUT PATH"
dtm <- rast(paste0(pth, "dtm.tif"))
tVis <- makeTerrainVisTorch(dtm,
cellSize=1,
innerRadius=2,
outerRadius=5,
hsRadius=50,
writeRaster=TRUE,
outName=paste0(pth, "tVisTorch.tif"),
device="cuda")

## End(Not run)
```

makeTPI *makeTPI*

Description

Calculate a topographic position index (TPI) from a digital terrain model (DTM) using torch

Usage

```
makeTPI(
  dtm,
  cellSize = 1,
  innerRadius = 2,
  outerRadius = 10,
  mode = "circle",
  writeRaster = FALSE,
  outName,
  device = "cpu"
)
```

Arguments

dtm	Input SpatRaster object representing bare earth surface elevations.
cellSize	Resolution of raster grid. Default is 1 m.
innerRadius	= Inner radius when using annulus moving window. Default is 2 cells.
outerRadius	= Outer radius when using a circle or annulus moving window. Default is 10 cells.
mode	Either "circle" or "annulus". Default is "circle".
writeRaster	TRUE or FALSE. Save output to disk. Default is TRUE.
outName	Name of output raster with full file path and extension.
device	"cpu" or "cuda". Use "cuda" for GPU computation. Without using the GPU, implementation will not be significantly faster than using non-tensor-based computation. Defaults is "cpu".

Details

Calculate a topographic position index (TPI) from a digital terrain model (DTM) using torch. TPI is calculated as the center cell elevation minus the mean elevation in a local moving window. Large, positive values indicate local topographic high points while large, negative values indicate topographic low points. Near zero values indicate flat areas. Larger windows can characterize the hillslope position of a cell while smaller windows can capture local topographic variability. Radii are specified using cell counts as opposed to map distance.

Value

Single-band SpatRaster object. If writeRaster = TRUE, the raster is also written to disk.

Examples

```
## Not run:
pth <- "OUTPUT PATH"
dtm <- rast(paste0(pth, "dtm.tif"))
tpiA3_11 <- makeTPI(dtm,cellSize=1,
innerRadius=3,
outerRadius=11,
mode="circle",
writeRaster=TRUE,
outName=paste0(pth, "tpiA3_11.tif"),
device="cuda")

## End(Not run)
```

---

makeTRI *makeTRI*

Description

Calculate a topographic roughness index (TRI) from a digital terrain model (DTM) using torch

Usage

```
makeTRI(
  dtm,
  cellSize = 1,
  roughRadius = 7,
  writeRaster = FALSE,
  outName,
  device = "cpu"
)
```

Arguments

dtm	Input SpatRaster object representing bare earth surface elevations.
cellSize	Resolution of raster grid. Default is 1 m.
roughRadius	radius of circular moving window. Default is 7 cells.
writeRaster	TRUE or FALSE. Save output to disk. Default is TRUE.
outName	Name of output raster with full file path and extension.
device	"cpu" or "cuda". Use "cuda" for GPU computation. Without using the GPU, implementation will not be significantly faster than using non-tensor-based computation. Defaults is "cpu".

Details

Calculate a topographic roughness index (TPI) from a digital terrain model (DTM) using torch. Us calculated as the square root of the standard deviation of slope in a local moving window. Output can be noisy. Radii are specified using cell counts as opposed to map distance.

Value

Single-band SpatRaster object. If writeRaster = TRUE, the raster is also written to disk.

Examples

```
## Not run:
pth <- "OUTPUT PATH"
dtm <- rast(paste0(pth, "dtm.tif"))
tri11 <- makeTRI(dtm,
cellSize=1,
roughRadius=11,
writeRaster=TRUE,
outName=paste0(pth, "tri11f.tif"),
device="cuda")

## End(Not run)
```

`make_ds_weights` *make_ds_weights*

Description

Create a shared mutable weight container for deep supervision loss weighting.

Usage

```
make_ds_weights(weights)
```

Arguments

`weights`	Numeric vector of initial loss weights. The first element is the primary (full-resolution) output; subsequent elements are ancillary outputs in coarsest-to-finest order. All values must be non-negative.

Details

Creates an R environment that holds the current per-output loss weights for deep supervision training. Pass the returned object to both `defineUnifiedFocalLossDS` (via the `weight_env` argument) and `callback_ds_weight_decay` so the callback can update the weights the loss function reads each epoch.

Value

An R environment with a single element `$values` containing the current weight vector. Passed by reference, so updates made by `callback_ds_weight_decay` are immediately visible to `ds_loss`.

plotTrainingSchedule *plotTrainingSchedule*

Description

Visualise per-stage base learning rates, an optional LR scheduler trace, and deep supervision weight decay without running a training loop.

Usage

```
plotTrainingSchedule(
  stageLRs,
  epochs,
  initialWeights = NULL,
  decay = "linear",
  primary_only_frac = 0.1,
  stageNames = NULL,
  scheduler = NULL
)
```

Arguments

stageLRs	Numeric vector of per-stage base learning rates matching the stageLRs argument passed to the model. A single value produces a one-bar chart.
epochs	Total number of training epochs.
initialWeights	Optional numeric vector of initial deep supervision loss weights as passed to make_ds_weights. When NULL the DS weight panel is omitted.
decay	Character string, either "linear" (default) or "cosine", matching the decay argument passed to callback_ds_weight_decay.
primary_only_frac	Numeric value in (0, 1) specifying the fraction of total epochs reserved for primary-loss-only training at the end of the run, matching the primary_only_frac argument passed to callback_ds_weight_decay. Defaults to 0.1.
stageNames	Optional character vector of stage labels for the learning rate chart. Defaults to "Stage 1", "Stage 2", ...
scheduler	Optional named list specifying an LR scheduler to visualise. Must contain at least type ("cosine_annealing", "one_cycle", or "step"). See Details for per-type parameters.

Details

Produces up to three panels depending on inputs. The left panel always shows a bar chart of per-stage base learning rates. When scheduler is supplied a centre panel shows the effective learning rate curve over all epochs. When initialWeights is supplied a right panel shows how each deep supervision weight evolves over epochs, replicating the exact decay applied by callback_ds_weight_decay.

The scheduler list accepts three types, each mirroring a torch R scheduler:

"cosine_annealing" Mirrors lr_cosine_annealing. Named elements: base_lr (starting LR; defaults to max(stageLRs)), T_max (epochs per half-cycle; defaults to epochs), eta_min (minimum LR, default 0). The LR oscillates between base_lr and eta_min with period 2 * T_max.

"one_cycle" Mirrors lr_one_cycle. Named elements: max_lr (peak LR; defaults to max(stageLRs)), pct_start (warmup fraction, default 0.3), div_factor (initial_lr = max_lr / div_factor, default 25), final_div_factor (min_lr = initial_lr / final_div_factor, default 1e4), anneal_strategy ("cos" or "linear", default "cos").

"step" Mirrors lr_step. Named elements: base_lr (starting LR; defaults to max(stageLRs)), step_size (epochs between LR drops; defaults to max(1, floor(epochs / 10))), gamma (multiplicative decay factor per drop, default 0.1).

Value

Invisibly returns a named list with element lr (a data frame of stage names and learning rates), lr_trace (a data frame of epoch-by-LR values when scheduler is supplied), and weights (a data frame of epoch-by-weight values when initialWeights is supplied).

---

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,
  doPad = FALSE,
  predPad = 0,
  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	instantiated model object as nn_module subclass as opposed to luz fitted object. Make sure to place model in evaluation mode.
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.
doPad	TRUE or FALSE. Whehter or not to add padding to chips. This is only necessary when using defineTerrainSeg(). Default is FALSE.
predPad	Only needed when using defineTerrainSeg() and doPad = TRUE. Should be the same as tCrop parameter in defineTerrainSeg(). Default is 0.
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)
```

`saveDynamicChips` *saveDynamicChips*

Description

Save chips meant to be generated dynamically to disk.

Usage

```
saveDynamicChips(
  chipsSF,
  chipSize = 512,
  cellSize = 1,
  outDir,
  mode = "All",
  useExistingDir = FALSE
)
```

Arguments

`chipsSF`	output from makeDynamicChipsSF().
`chipSize`	size of chips to generate. Default is 512 (512-by-512 cells).
`cellSize`	cell size of input and output data. Default is 1 m.
`outDir`	full or relative path to output directory. Must include final forward slash in path.
`mode`	either "All", "Positive", or "Divided". If "All", all chips and masks are saved. If "Positive", only chips and masks containing positive cells are maintained. If "Divided", background-only and positive-containing chips are saved but written to separate directories. For multiclass, use "All". Default is "All".
`useExistingDir`	TRUE or FALSE. Whether or not to use a directory that already contains chips. Default is FALSE.

Details

Save chips defined by makeDynamicChipsSF() to disk. It is not required to save dynamically generated chips to disk. This is primarily a utility function.

Value

chips and masks written to disk. No R object is returned.

setSeed *setSeed*

Description

Set a global random seed for reproducible geodl workflows.

Usage

```
setSeed(seed)
```

Arguments

seed	Single integer random seed.

Details

A geodl/torch training pipeline draws from two independent random number generator (RNG) streams: R's base RNG and torch's own RNG. `setSeed()` seeds both from a single value so that dataset shuffling, data augmentation, model weight initialization, and the training loop all produce identical results across runs.

The following sources of randomness are covered:

Dataset shuffling `torch::dataloader(..., shuffle = TRUE)` and any `dplyr::sample_n()`-based shuffling (e.g. `makeChipsDF(shuffle = TRUE)`) read from R's base RNG.

Data augmentation `defineSegDataSet()` and `defineDynamicSegDataSet()` choose which augmentations to apply and their magnitudes with base R functions (`runif()`, `sample()`, `dplyr::sample_n()`), which read from R's base RNG.

Model weight initialization All `defineXXX` model constructors (e.g. `defineUNet`, `defineDLab`) rely on torch's default layer initializers, which draw from torch's RNG.

Training loop Any stochastic operations inside `luz::fit()` (e.g. dropout, if present in a given architecture) draw from torch's RNG.

Call this before creating the dataset, dataloader, and model, and before calling `luz::fit()` — any random draws that occur before seeding will not be reproducible.

When `torch::dataloader()` is used with `num_workers > 0`, torch automatically reseeds each worker subprocess (with both `set.seed()` and `torch::torch_manual_seed()`) from the parent process's RNG state at the time the workers are spawned. As long as `setSeed()` is called once before the dataloader is created, and `num_workers` is held constant between runs, worker seeding is deterministic and requires no further action.

Full determinism on GPU is still subject to caveats outside this function's control: some cuDNN convolution algorithms are inherently non-deterministic regardless of RNG seeding, and torch's R bindings do not currently expose a "deterministic algorithms" toggle. For strict bit-for-bit reproducibility, training on CPU remains the most reliable option.

Value

Invisibly returns seed (as an integer).

---

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,
  cCodes,
  cNames,
  cColors,
  padding = 10
)
```

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.
cCodes	class indices as a vector of integer values equal in length to the number of classes.
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.
padding	to place between chips in plot. Default is 10.

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,
          cCodes=c(1,2),
          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,
  padding = 10,
  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.
padding	Number of cells of padding to add around each example in resulting image.
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,
  cCodes,
  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 blue 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.
cCodes	class indices as a vector of integer values equal in length to the number of classes.
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,
          cCodes=c(0,1,2,3,4),
          cNames=c("Background",
                   "Building",
                   "Woodland",
                   "Water",
                   "Road"),
          cColor=c("gray",
                   "darksalmon",
                    "forestgreen",
                    "lightblue",
                    "black"),
          useSeed = FALSE,
          seed = 42)

## End(Not run)
```

---

viewDist *viewDist*

Description

Visualise a mini-batch of image chips, masks, distance-to-boundary grids, and the resulting edge-weight surface for a DataLoader that supplies distance grids.

Usage

```
viewDist(
  dataLoader,
  nCols = 3,
  r = 1,
  g = 2,
  b = 3,
  cCodes,
  cNames,
  cColors,
```

```
  edgeMode = "gaussian",
  distThreshold = 10,
  minWght = 0.5,
  maxWght = 1,
  prioritizeEdge = TRUE,
  padding = 10
)
```

Arguments

dataLoader	Instantiated DataLoader created with `torch::dataloader()`. The batch must contain named elements `image`, `mask`, and `dist`.
nCols	Number of chip columns in each image grid. Default is 3.
r	Index of the image channel mapped to the red display channel. Default is 1. For single-band imagery, assign the same index to `r`, `g`, and `b`.
g	Index of the image channel mapped to the green display channel. Default is 2.
b	Index of the image channel mapped to the blue display channel. Default is 3.
cCodes	Integer class codes as a vector, one per class.
cNames	Character vector of class names, in the same order as `cCodes`.
cColors	Character vector of display colours for the mask panel, in the same order as `cCodes`.
edgeMode	Character string specifying how distances are converted to weights. One of `"threshold"`, `"linear"`, or `"gaussian"` (default). Must match the `edgeMode` used in `defineUnifiedFocalLoss`.
distThreshold	Positive numeric. The distance value (in the same units as the distance raster) that defines the boundary of the high-weight zone. Default is 10.
minWght	Numeric. Minimum per-pixel weight. Must be less than `maxWght`. Default is 0.5.
maxWght	Numeric. Maximum per-pixel weight. Must be greater than `minWght`. Default is 1.0.
prioritizeEdge	Logical. When `TRUE` (default), pixels near class boundaries (small distance) receive `maxWght` and pixels far from boundaries receive `minWght`. When `FALSE`, the relationship is reversed.
padding	Integer. Number of pixels of padding between chips in the grid. Default is 10.

Details

Plots four panels stacked vertically:

1. RGB image chips arranged in a grid.
2. Corresponding class masks.
3. Distance-to-class-boundary grids produced by `makeBoundaryDistance`.
4. The per-pixel edge-weight surface derived from those distances, computed with the same formula used by `defineUnifiedFocalLoss` when `useEdgeDist = TRUE`.

The weight surface mirrors the three modes available in `defineUnifiedFocalLoss`:

`"threshold"` Pixels within `distThreshold` receive one weight (`maxWght` when `prioritizeEdge = TRUE`); all others receive the opposite weight.

`"linear"` Weight decreases linearly from `maxWght` at distance 0 to `minWght` at `distThreshold` (when `prioritizeEdge = TRUE`), clamped beyond the threshold.

`"gaussian"` Weight follows a Gaussian curve centred at distance 0 with sigma = distThreshold / 3, so the weight equals approximately `minWght` at the threshold and is clamped to `minWght` beyond it (when `prioritizeEdge = TRUE`).

The DataLoader must return batches with three named elements: `image`, `mask`, and `dist`. The `dist` element should be a tensor of shape (N, 1, H, W) containing the per-pixel distances produced by `makeBoundaryDistance`.

Value

Called for its side effect (plots to the active graphics device). Returns `invisible(NULL)`.

Examples

```
## Not run:
viewDist(
  dataLoader    = trainDL,
  nCols         = 4,
  r = 1, g = 2, b = 3,
  cCodes        = c(1, 2, 3, 4, 5),
  cNames        = c("Background", "Forest", "Water", "Urban", "Agriculture"),
  cColors       = c("gray", "darkgreen", "blue", "red", "yellow"),
  edgeMode      = "gaussian",
  distThreshold = 10,
  minWght       = 0.4,
  maxWght       = 1.0,
  prioritizeEdge = TRUE
)

## End(Not run)
```

Index

addCoordNoise, 3
assessDL, 4
assessPnts, 6
assessRaster, 8

callback_ds_weight_decay, 10
callback_ema, 11
callback_freeze_stages, 12
callback_per_stage_optimizer, 13
callback_reconfigure_loss, 14
callback_save_model_state_dict, 16
callback_training_log, 16
callback_unfreeze_stages, 17
checkDynamicChips, 18
countParams, 19

defineDLab, 20
defineDynamicSegDataSet, 22
defineDynmamicSegDataSet
(defineDynamicSegDataSet), 22
defineEfficientUNetB2, 25
defineMobileUNet, 27, *31*
defineRegressionLoss, 28
defineResNetUNet, 30
defineSegDataSet, 32
defineTerrainSeg, 35
defineUNet, 37, *41*
defineUNet3p, 40
defineUnifiedFocalLoss, 42
defineUnifiedFocalLossDS, 46
describeBatch, 49
describeChips, 50

luz_metric_f1score, 51
luz_metric_overall_accuracy, 53
luz_metric_precision, 55
luz_metric_recall, 57

make_ds_weights, 79
makeAspect, 59
makeBoundaryDistance, 60
makeChips, 61, *65, 66*
makeChipsDF, 63
makeChipsMultiClass, 65
makeCrv, 66
makeDynamicChipsSF, 68
makeHillshade, 69
makeMasks, 71
makeSlope, 73
makeTerrainVisTerra, 74
makeTerrainVisTorch, 75
makeTPI, 77
makeTRI, 78

plotTrainingSchedule, 80
predictSpatial, 81

saveDynamicChips, 84
setSeed, 85

viewBatch, 86
viewBatchPreds, 87
viewChips, 89
viewDist, 91