DeepDIVA Documentation

Vinaychandran Pondenkandath and Michele Alberti

Sep 26, 2020

CONTENTS:

sphinx-quickstart on Mon Jul 9 11:19:11 2018. You can adapt this file completely to your liking, but it should at least contain the root *toctree* directive.

CHAPTER

ONE

DATASETS PACKAGE

1.1 Submodules

1.2 datasets.bidimensional_dataset module

Load a dataset of bidimensional points by specifying the folder where its located.

class datasets.bidimensional_dataset.**Bidimensional**(*path*, *transform=None*, *target_transform=None*)

Bases: torch.utils.data.dataset.Dataset

This class loads the data.csv file and prepares it as a dataset.

datasets.bidimensional_dataset.**load_dataset**(*dataset_folder*)

Loads the dataset from file system and provides the dataset splits for train validation and test

The dataset is expected to be in the following structure, where 'dataset_folder' has to point to the root of the three folder train/val/test.

Example:

dataset_folder = "~/../../data/bd_xor"

which contains the splits sub-folders as follow:

'dataset_folder'/train 'dataset_folder'/val 'dataset_folder'/test

Parameters **dataset_folder** (*string*) – Path to the dataset on the file System

Returns

- **train_ds** (*data.Dataset*)
- **val_ds** (*data.Dataset*)
- **test_ds** (*data.Dataset*) – Train, validation and test splits

1.3 datasets.image_folder_dataset module

Load a dataset of images by specifying the folder where its located.

class datasets.image_folder_dataset.**ImageFolderApply**(*path*, *transform=None*, *target_transform=None*, *classify=False*)

Bases: torch.utils.data.dataset.Dataset

TODO fill me

class datasets.image_folder_dataset.**ImageFolderInMemory**(*path*, *transform=None*, *target_transform=None*, *workers=1*)

Bases: torch.utils.data.dataset.Dataset

This class loads the data provided and stores it entirely in memory as a dataset.

It makes use of torchvision.datasets.ImageFolder() to create a dataset. Afterward all images are sequentially stored in memory for faster use when paired with dataloders. It is responsibility of the user ensuring that the dataset actually fits in memory.

datasets.image_folder_dataset.**load_dataset**(*dataset_folder*, *in_memory=False*, *workers=1*)

Loads the dataset from file system and provides the dataset splits for train validation and test

The dataset is expected to be in the following structure, where 'dataset_folder' has to point to the root of the three folder train/val/test.

Example:

dataset_folder = "~/../../data/cifar"

which contains the splits sub-folders as follow:

'dataset_folder'/train 'dataset_folder'/val 'dataset_folder'/test

In each of the three splits (train, val, test) should have different classes in a separate folder with the class name. The file name can be arbitrary i.e. it does not have to be 0-* for classes 0 of MNIST.

Example:

train/dog/whatever.png train/dog/you.png train/dog/like.png

train/cat/123.png train/cat/nsdf3.png train/cat/**asd932_**.png

train/"class_name"/*.png

Parameters

- **dataset_folder** (*string*) – Path to the dataset on the file System
- **in_memory** (*boolean*) – Load the whole dataset in memory. If False, only file names are stored and images are loaded on demand. This is slower than storing everything in memory.
- **workers** (*int*) – Number of workers to use for the dataloaders

Returns

- **train_ds** (*data.Dataset*)
- **val_ds** (*data.Dataset*)
- **test_ds** (*data.Dataset*) – Train, validation and test splits

1.4 datasets.image_folder_triplet module

Load a dataset of images by specifying the folder where its located and prepares it for triplet similarity matching training.

class datasets.image_folder_triplet.**ImageFolderTriplet**(*path*, *train=None*, *num_triplets=None*, *in_memory=None*, *transform=None*, *target_transform=None*, *workers=None*)

Bases: torch.utils.data.dataset.Dataset

This class loads the data provided and stores it entirely in memory as a dataset. Additionally, triplets will be generated in the format of [a, p, n] and their file names stored in memory.

generate_triplets()

Generate triplets for training. Triplets have format [anchor, positive, negative]

datasets.image_folder_triplet.**load_dataset**(*dataset_folder*, *num_triplets=None*, *in_memory=False*, *workers=1*, *only_evaluate=False*)

Loads the dataset from file system and provides the dataset splits for train validation and test.

The dataset is expected to be in the same structure as described in image_folder_dataset.load_dataset()

Parameters

- **dataset_folder** (*string*) – Path to the dataset on the file System
- **num_triplets** (*int*) – Number of triplets [a, p, n] to generate on dataset creation
- **in_memory** (*boolean*) – Load the whole dataset in memory. If False, only file names are stored and images are loaded on demand. This is slower than storing everything in memory.
- **workers** (*int*) – Number of workers to use for the dataloaders
- **only_evaluate** (*boolean*) – Flag : if True, only the test set is loaded.

Returns

- **train_ds** (*data.Dataset*)
- **val_ds** (*data.Dataset*)
- **test_ds** (*data.Dataset*) – Train, validation and test splits

1.5 datasets.multi_label_image_folder_dataset module

Load a dataset of images by specifying the folder where its located.

class datasets.multi_label_image_folder_dataset.**MultiLabelImageFolder**(*path*, *transform=None*, *target_transform=None*, *workers=1*)

Bases: torch.utils.data.dataset.Dataset

This class loads the multi-label image data provided.

datasets.multi_label_image_folder_dataset.**load_dataset**(*dataset_folder*, *in_memory=False*, *workers=1*)

Loads the dataset from file system and provides the dataset splits for train validation and test

The dataset is expected to be in the following structure, where 'dataset_folder' has to point to the root of the three folder train/val/test.

Example:

dataset_folder = "~/../../data/dataset_folder"

which contains the splits sub-folders as follow:

'dataset_folder'/train 'dataset_folder'/val 'dataset_folder'/test

Each of the three splits (train, val, test) should contain a folder called 'images' containing all of the images (the file names of the images can be arbitrary). The split folder should also contain a csv file called 'labels.csv' formatted so:

filename,class_0,class_1,. . . ,class_n images/img_1.png,1,-1,-1,. . . ,1

where the filename is the relative path to the image file from the split folder and 1/-1 to indicate presence/absence of a particular label.

Example:

train/image/whatever.png train/image/you.png train/image/like.png train/labels.csv

and the labels.csv would contain:

filename,cat,dog,elephant image/whatever.png,1,1,-1 image/you.png,1,-1,-1 image/like.png,-1,1,1

Parameters

- **dataset_folder** (*string*) – Path to the dataset on the file System
- **in_memory** (*boolean*) – Load the whole dataset in memory. If False, only file names are stored and images are loaded on demand. This is slower than storing everything in memory.
- **workers** (*int*) – Number of workers to use for the dataloaders

Returns

- **train_ds** (*data.Dataset*)
- **val_ds** (*data.Dataset*)
- **test_ds** (*data.Dataset*) – Train, validation and test splits

1.6 Module contents

CHAPTER

TWO

MODELS PACKAGE

2.1 Submodules

2.2 models.registry module

Credit for the initial implementation of the @Model decorating system to Narayan Schuetz, University of Bern

models.registry.**Model**(**args*, ***kwargs*)

Decorator function, makes model definition a bit more obvious than relying on python's underscore variant

2.3 Module contents

CHAPTER

THREE

TEMPLATE PACKAGE

3.1 Subpackages

3.1.1 template.runner package

3.1.1.1 Subpackages

template.runner.apply_model package

Submodules

template.runner.apply_model.apply_model module

This file is the template for the boilerplate of train/test of a DNN

There are a lot of parameter which can be specified to modify the behaviour and they should be used instead of hard-coding stuff.

@authors: Vinaychandran Pondenkandath , Michele Alberti

class template.runner.apply_model.apply_model.**ApplyModel**

Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *lr*, *output_channels*, *classify*, ***kwargs*)

This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **lr** (*float*) – Value for learning rate
- **kwargs** (*dict*) – Any additional arguments.
- **output_channels** (*int*) – Specify shape of final layer of network.
- **classify** (*boolean*) – Specifies whether to generate a classification report for the data or not.

Returns **None** – None

Return type None

template.runner.apply_model.evaluate module

template.runner.apply_model.evaluate.**feature_extract**(*data_loader*, *model*, *writer*, *epoch*, *no_cuda*, *log_interval*, *classify*, ***kwargs*)

The evaluation routine

Parameters

- **data_loader** (*torch.utils.data.DataLoader*) – The dataloader of the evaluation set
- **model** (*torch.nn.module*) – The network model being used
- **writer** (*tensorboardX.writer.SummaryWriter*) – The tensorboard writer object. Used to log values on file for the tensorboard visualization.
- **epoch** (*int*) – Number of the epoch (for logging purposes)
- **no_cuda** (*boolean*) – Specifies whether the GPU should be used or not. A value of 'True' means the CPU will be used.
- **log_interval** (*int*) – Interval limiting the logging of mini-batches. Default value of 10.
- **classify** (*boolean*) – Specifies whether to generate a classification report for the data or not.

Returns

Return type None

template.runner.apply_model.setup module

template.runner.apply_model.setup.**set_up_dataloader**(*model_expected_input_size*, *dataset_folder*, *batch_size*, *workers*, *inmem*, *multi_crop*, *classify*, ***kwargs*)

Set up the dataloaders for the specified datasets.

Parameters

- *param model_expected_input_size: tuple* – Specify the height and width that the model expects.
- *param dataset_folder: string* – Path string that points to the three folder train/val/test. Example: ~/../../data/svhn
- *param batch_size: int* – Number of datapoints to process at once
- *param workers: int* – Number of workers to use for the dataloaders
- *param inmem: boolean* – Flag: if False, the dataset is loaded in an online fashion i.e. only file names are stored and images are loaded on demand. This is slower than storing everything in memory.
- *param multi_crop: int* – if None, the MultiCrop transform is not applied to the data. Otherwise, multi_crop contains an integer which specifies how many crops to make from each image.
- *param classify : boolean* – Specifies whether to generate a classification report for the data or not.

- *param kwargs: dict* – Any additional arguments.
- *return: dataloader, dataloader, dataloader, int* – Three dataloaders for train, val and test. Number of classes for the model.

Module contents

class template.runner.apply_model.**ApplyModel**
Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *lr*, *output_channels*, *classify*, ***kwargs*)
This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **lr** (*float*) – Value for learning rate
- **kwargs** (*dict*) – Any additional arguments.
- **output_channels** (*int*) – Specify shape of final layer of network.
- **classify** (*boolean*) – Specifies whether to generate a classification report for the data or not.

Returns **None** – None

Return type None

template.runner.bidimensional package

Submodules

template.runner.bidimensional.bidimensional module

This file is the template for the boilerplate of train/test of a DNN on a bidimensional dataset In particular, it is designed to work with clouds of bi-dimensional points.

class template.runner.bidimensional.bidimensional.**Bidimensional**
Bases: *template.runner.image_classification.image_classification.ImageClassification*

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *validation_interval*, *checkpoint_all_epochs*, ***kwargs*)
This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard.SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model

- **epochs** (*int*) – Number of epochs to train
- **lr** (*float*) – Value for learning rate
- **kwargs** (*dict*) – Any additional arguments.
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split
- **test_value** (*float*) – Accuracy value for test split

Module contents

class template.runner.bidimensional.**Bidimensional**

Bases: *template.runner.image_classification.image_classification.ImageClassification*

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *validation_interval*, *checkpoint_all_epochs*, ***kwargs*)

This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard.SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **epochs** (*int*) – Number of epochs to train
- **lr** (*float*) – Value for learning rate
- **kwargs** (*dict*) – Any additional arguments.
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split
- **test_value** (*float*) – Accuracy value for test split

template.runner.image_classification package

Submodules

template.runner.image_classification.evaluate module

template.runner.image_classification.evaluate.**evaluate**(*data_loader*, *model*, *criterion*, *writer*, *epoch*, *logging_label*, *no_cuda=False*, *log_interval=20*, ***kwargs*)

The evaluation routine

Parameters

- **data_loader** (*torch.utils.data.DataLoader*) – The dataloader of the evaluation set
- **model** (*torch.nn.module*) – The network model being used
- **criterion** (*torch.nn.loss*) – The loss function used to compute the loss of the model
- **writer** (*tensorboardX.writer.SummaryWriter*) – The tensorboard writer object. Used to log values on file for the tensorboard visualization.
- **epoch** (*int*) – Number of the epoch (for logging purposes)
- **logging_label** (*string*) – Label for logging purposes. Typically 'test' or 'valid'. Its prepended to the logging output path and messages.
- **no_cuda** (*boolean*) – Specifies whether the GPU should be used or not. A value of 'True' means the CPU will be used.
- **log_interval** (*int*) – Interval limiting the logging of mini-batches. Default value of 10.

Returns **top1.avg** – Accuracy of the model of the evaluated split

Return type float

template.runner.image_classification.image_classification module

This file is the template for the boilerplate of train/test of a DNN for image classification

There are a lot of parameter which can be specified to modify the behaviour and they should be used instead of hard-coding stuff.

class template.runner.image_classification.image_classification.**ImageClassification**

Bases: object

classmethod prepare(*model_name*, ***kwargs*)

Loads and prepares the data, the optimizer and the criterion

Parameters

- **model_name** (*str*) – Name of the model. Used for loading the model.
- **kwargs** (*dict*) – Any additional arguments.

Returns

- **model** (*DataParallel*) – The model to train
- **num_classes** (*int*) – How many different classes there are in our problem. Used for loading the model.

- **best_value** (*float*) – Best value of the model so far. Non-zero only in case of –resume being used
- **train_loader** (*torch.utils.data.dataloader.DataLoader*) – Training dataloader
- **val_loader** (*torch.utils.data.dataloader.DataLoader*) – Validation dataloader
- **test_loader** (*torch.utils.data.dataloader.DataLoader*) – Test set dataloader
- **optimizer** (*torch.optim*) – Optimizer to use during training, e.g. SGD
- **criterion** (*torch.nn.modules.loss*) – Loss function to use, e.g. cross-entropy

classmethod single_run(***kwargs*)

This is the main routine where train(), validate() and test() are called.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split
- **test_value** (*float*) – Accuracy value for test split

classmethod test_routine(*model_name*, *num_classes*, *criterion*, *epochs*, *current_log_folder*, *writer*, ***kwargs*)

Load the best model according to the validation score (early stopping) and runs the test routine.

Parameters

- **model_name** (*str*) – name of the model. Used for loading the model.
- **num_classes** (*int*) – How many different classes there are in our problem. Used for loading the model.
- **criterion** (*torch.nn.modules.loss*) – Loss function to use, e.g. cross-entropy
- **epochs** (*int*) – After how many epochs are we testing
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **writer** (*Tensorboard.SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **kwargs** (*dict*) – Any additional arguments.

Returns **test_value** – Accuracy value for test split

Return type float

classmethod train_routine(*best_value*, *decay_lr*, *validation_interval*, *start_epoch*, *epochs*, *checkpoint_all_epochs*, *current_log_folder*, ***kwargs*)

Performs the training and validatation routines

Parameters

- **best_value** (*float*) – Best value of the model so far. Non-zero only in case of –resume being used
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **start_epoch** (*int*) – Int to initialize the starting epoch. Non-zero only in case of –resume being used
- **epochs** (*int*) – Number of epochs to train
- **checkpoint_all_epochs** (*bool*) – Save checkpoint at each epoch

- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **kwargs** (*dict*) – Any additional arguments.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split

template.runner.image_classification.train module

template.runner.image_classification.train.**train**(*train_loader*, *model*, *criterion*, *optimizer*, *writer*, *epoch*, *no_cuda=False*, *log_interval=25*, ***kwargs*)

Training routine

Parameters

- **train_loader** (*torch.utils.data.DataLoader*) – The dataloader of the train set.
- **model** (*torch.nn.module*) – The network model being used.
- **criterion** (*torch.nn.loss*) – The loss function used to compute the loss of the model.
- **optimizer** (*torch.optim*) – The optimizer used to perform the weight update.
- **writer** (*tensorboardX.writer.SummaryWriter*) – The tensorboard writer object. Used to log values on file for the tensorboard visualization.
- **epoch** (*int*) – Number of the epoch (for logging purposes).
- **no_cuda** (*boolean*) – Specifies whether the GPU should be used or not. A value of 'True' means the CPU will be used.
- **log_interval** (*int*) – Interval limiting the logging of mini-batches. Default value of 10.

Returns **top1.avg** – Accuracy of the model of the evaluated split

Return type float

template.runner.image_classification.train.**train_one_mini_batch**(*model*, *criterion*, *optimizer*, *input*, *target*, *loss_meter*, *acc_meter*)

This routing train the model passed as parameter for one mini-batch

Parameters

- **model** (*torch.nn.module*) – The network model being used.
- **criterion** (*torch.nn.loss*) – The loss function used to compute the loss of the model.
- **optimizer** (*torch.optim*) – The optimizer used to perform the weight update.
- **input** (*torch.autograd.Variable*) – The input data for the mini-batch
- **target** (*torch.autograd.Variable*) – The target data (labels) for the mini-batch
- **loss_meter** (*AverageMeter*) – Tracker for the overall loss
- **acc_meter** (*AverageMeter*) – Tracker for the overall accuracy

Returns

- **acc** (*float*) – Accuracy for this mini-batch
- **loss** (*float*) – Loss for this mini-batch

Module contents

class template.runner.image_classification.**ImageClassification**
Bases: object

classmethod prepare(*model_name*, ***kwargs*)
Loads and prepares the data, the optimizer and the criterion

Parameters

- **model_name** (*str*) – Name of the model. Used for loading the model.
- **kwargs** (*dict*) – Any additional arguments.

Returns

- **model** (*DataParallel*) – The model to train
- **num_classes** (*int*) – How many different classes there are in our problem. Used for loading the model.
- **best_value** (*float*) – Best value of the model so far. Non-zero only in case of –resume being used
- **train_loader** (*torch.utils.data.dataloader.DataLoader*) – Training dataloader
- **val_loader** (*torch.utils.data.dataloader.DataLoader*) – Validation dataloader
- **test_loader** (*torch.utils.data.dataloader.DataLoader*) – Test set dataloader
- **optimizer** (*torch.optim*) – Optimizer to use during training, e.g. SGD
- **criterion** (*torch.nn.modules.loss*) – Loss function to use, e.g. cross-entropy

classmethod single_run(***kwargs*)
This is the main routine where train(), validate() and test() are called.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split
- **test_value** (*float*) – Accuracy value for test split

classmethod test_routine(*model_name*, *num_classes*, *criterion*, *epochs*, *current_log_folder*, *writer*, ***kwargs*)
Load the best model according to the validation score (early stopping) and runs the test routine.

Parameters

- **model_name** (*str*) – name of the model. Used for loading the model.
- **num_classes** (*int*) – How many different classes there are in our problem. Used for loading the model.
- **criterion** (*torch.nn.modules.loss*) – Loss function to use, e.g. cross-entropy
- **epochs** (*int*) – After how many epochs are we testing
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved

- **writer** (*Tensorboard.SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **kwargs** (*dict*) – Any additional arguments.

Returns **test_value** – Accuracy value for test split

Return type float

classmethod train_routine(*best_value*, *decay_lr*, *validation_interval*, *start_epoch*, *epochs*, *checkpoint_all_epochs*, *current_log_folder*, ***kwargs*)

Performs the training and validatation routines

Parameters

- **best_value** (*float*) – Best value of the model so far. Non-zero only in case of –resume being used
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **start_epoch** (*int*) – Int to initialize the starting epoch. Non-zero only in case of –resume being used
- **epochs** (*int*) – Number of epochs to train
- **checkpoint_all_epochs** (*bool*) – Save checkpoint at each epoch
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **kwargs** (*dict*) – Any additional arguments.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split

template.runner.multi_label_image_classification package

Submodules

template.runner.multi_label_image_classification.evaluate module

template.runner.multi_label_image_classification.evaluate.**compute_jss**(*target*, *preds*)

template.runner.multi_label_image_classification.evaluate.**get_preds_from_minibatch**(*minibatch*)

template.runner.multi_label_image_classification.evaluate.**jaccard_similarity_score**(*targets*, *preds*)

template.runner.multi_label_image_classification.evaluate.**test**(*test_loader*, *model*, *criterion*, *writer*, *epoch*, *no_cuda=False*, *log_interval=20*, ***kwargs*)

Wrapper for _evaluate() with the intent to test the model

template.runner.multi_label_image_classification.evaluate.**validate**(*val_loader*, *model*, *criterion*, *writer*, *epoch*, *no_cuda=False*, *log_interval=20*, ***kwargs*)

Wrapper for _evaluate() with the intent to validate the model.

template.runner.multi_label_image_classification.multi_label_image_classification module

This file is the template for the boilerplate of train/test of a DNN for image classification

There are a lot of parameter which can be specified to modify the behaviour and they should be used instead of hard-coding stuff.

class template.runner.multi_label_image_classification.multi_label_image_classification.**Mu**

Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *validation_interval*, *checkpoint_all_epochs*, ***kwargs*)

This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard.SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **epochs** (*int*) – Number of epochs to train
- **lr** (*float*) – Value for learning rate
- **kwargs** (*dict*) – Any additional arguments.
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split
- **test_value** (*float*) – Accuracy value for test split

template.runner.multi_label_image_classification.setup module

template.runner.multi_label_image_classification.setup.**set_up_dataloaders**(*model_expected_input_s*, *dataset_folder*, *batch_size*, *workers*, *disable_dataset_integrity*, *enable_deep_dataset_integ*, *inmem=False*, ***kwargs*)

Set up the dataloaders for the specified datasets.

Parameters

- **model_expected_input_size** (*tuple*) – Specify the height and width that the model expects.
- **dataset_folder** (*string*) – Path string that points to the three folder train/val/test. Example: ~/../../data/svhn
- **batch_size** (*int*) – Number of datapoints to process at once
- **workers** (*int*) – Number of workers to use for the dataloaders
- **inmem** (*boolean*) – Flag: if False, the dataset is loaded in an online fashion i.e. only file names are stored and images are loaded on demand. This is slower than storing everything in memory.

Returns

- **train_loader** (*torch.utils.data.DataLoader*)
- **val_loader** (*torch.utils.data.DataLoader*)
- **test_loader** (*torch.utils.data.DataLoader*) – Dataloaders for train, val and test.
- *int* – Number of classes for the model.

template.runner.multi_label_image_classification.train module

template.runner.multi_label_image_classification.train.**compute_jss**(*target*, *preds*)

template.runner.multi_label_image_classification.train.**get_preds_from_minibatch**(*minibatch*)

template.runner.multi_label_image_classification.train.**jaccard_similarity_score**(*targets*, *preds*)

template.runner.multi_label_image_classification.train.**train**(*train_loader*, *model*, *criterion*, *optimizer*, *writer*, *epoch*, *no_cuda=False*, *log_interval=25*, ***kwargs*)

Training routine

Parameters

- **train_loader** (*torch.utils.data.DataLoader*) – The dataloader of the train set.
- **model** (*torch.nn.module*) – The network model being used.
- **criterion** (*torch.nn.loss*) – The loss function used to compute the loss of the model.
- **optimizer** (*torch.optim*) – The optimizer used to perform the weight update.
- **writer** (*tensorboardX.writer.SummaryWriter*) – The tensorboard writer object. Used to log values on file for the tensorboard visualization.
- **epoch** (*int*) – Number of the epoch (for logging purposes).
- **no_cuda** (*boolean*) – Specifies whether the GPU should be used or not. A value of 'True' means the CPU will be used.
- **log_interval** (*int*) – Interval limiting the logging of mini-batches. Default value of 10.

Returns **top1.avg** – Accuracy of the model of the evaluated split

Return type float

template.runner.multi_label_image_classification.train.**train_one_mini_batch**(*model*, *criterion*, *optimizer*, *input*, *target*, *loss_meter*, *jss_meter*)

This routing train the model passed as parameter for one mini-batch

Parameters

- **model** (*torch.nn.module*) – The network model being used.
- **criterion** (*torch.nn.loss*) – The loss function used to compute the loss of the model.
- **optimizer** (*torch.optim*) – The optimizer used to perform the weight update.
- **input** (*torch.autograd.Variable*) – The input data for the mini-batch
- **target** (*torch.autograd.Variable*) – The target data (labels) for the mini-batch
- **loss_meter** (*AverageMeter*) – Tracker for the overall loss
- **jss_meter** (*AverageMeter*) – Tracker for the overall Jaccard Similarity Score

Returns **loss** – Loss for this mini-batch

Return type float

Module contents

class template.runner.multi_label_image_classification.**MultiLabelImageClassification**
Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *validation_interval*, *checkpoint_all_epochs*, ***kwargs*)
This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard.SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **epochs** (*int*) – Number of epochs to train
- **lr** (*float*) – Value for learning rate
- **kwargs** (*dict*) – Any additional arguments.
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split
- **test_value** (*float*) – Accuracy value for test split

template.runner.process_activation package

Submodules

template.runner.process_activation.activation module

class template.runner.process_activation.activation.**Activation**(*log_folder*, *model_name*, *dataset*, *process_size*, *save_cover*, *no_cuda*)
Bases: object

add_epoch(*epoch_number*, *epoch_accuracy*, *model*)
This method collect, compute and save all activation data (and mean activation data) from a given epoch

Parameters

- **epoch_number** (*int*) – Epoch number of the processing.
- **epoch_accuracy** (*int*) – Epoch accuracy retrived by the last training.
- **model** (*Torch.nn.model*) – PyTorch model trained.

Returns

Return type None

init(*model*)
This method initialize internal global according to model passed. Storage and create custom folders on the disk.

Parameters **model** (*Torch.nn.model*) – PyTorch model initialized.

Returns

Return type None

resolve_items()
This method prepare all the items to process, prepare and save cover for items (if needed), create the model's shape internally.

Parameters **None**

Returns

Return type None

template.runner.process_activation.evaluate module

template.runner.process_activation.evaluate.**test**(*test_loader*, *model*, *criterion*, *writer*, *epoch*, *no_cuda=False*, *log_interval=20*, ***kwargs*)
Wrapper for _evaluate() with the intent to test the model

template.runner.process_activation.evaluate.**validate**(*val_loader*, *model*, *criterion*, *writer*, *epoch*, *no_cuda=False*, *log_interval=20*, ***kwargs*)
Wrapper for _evaluate() with the intent to validate the model.

template.runner.process_activation.process_activation module

This file is the template for the boilerplate of train/test of a DNN for image classification

There are a lot of parameter which can be specified to modify the behaviour and they should be used instead of hard-coding stuff.

class template.runner.process_activation.process_activation.**ProcessActivation**
Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *validation_interval*, *checkpoint_all_epochs*, ***kwargs*)
DESC

Parameters **Param** – Desc

Returns

Return type None

template.runner.process_activation.train module

template.runner.process_activation.train.**train**(*train_loader*, *model*, *criterion*, *optimizer*, *writer*, *epoch*, *no_cuda=False*, *log_interval=25*, ***kwargs*)

Training routine

Parameters

- **train_loader** (*torch.utils.data.DataLoader*) – The dataloader of the train set.
- **model** (*torch.nn.module*) – The network model being used.
- **criterion** (*torch.nn.loss*) – The loss function used to compute the loss of the model.
- **optimizer** (*torch.optim*) – The optimizer used to perform the weight update.
- **writer** (*tensorboardX.writer.SummaryWriter*) – The tensorboard writer object. Used to log values on file for the tensorboard visualization.
- **epoch** (*int*) – Number of the epoch (for logging purposes).
- **no_cuda** (*boolean*) – Specifies whether the GPU should be used or not. A value of 'True' means the CPU will be used.
- **log_interval** (*int*) – Interval limiting the logging of mini-batches. Default value of 10.

Returns **top1.avg** – Accuracy of the model of the evaluated split

Return type float

template.runner.process_activation.train.**train_one_mini_batch**(*model*, *criterion*, *optimizer*, *input_var*, *target_var*, *loss_meter*, *acc_meter*)

This routing train the model passed as parameter for one mini-batch

Parameters

- **model** (*torch.nn.module*) – The network model being used.
- **criterion** (*torch.nn.loss*) – The loss function used to compute the loss of the model.
- **optimizer** (*torch.optim*) – The optimizer used to perform the weight update.
- **input_var** (*torch.autograd.Variable*) – The input data for the mini-batch
- **target_var** (*torch.autograd.Variable*) – The target data (labels) for the mini-batch
- **loss_meter** (*AverageMeter*) – Tracker for the overall loss
- **acc_meter** (*AverageMeter*) – Tracker for the overall accuracy

Returns

- **acc** (*float*) – Accuracy for this mini-batch
- **loss** (*float*) – Loss for this mini-batch

Module contents

class template.runner.process_activation.**ProcessActivation**
Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *validation_interval*, *checkpoint_all_epochs*, ***kwargs*)
DESC

Parameters **Param** – Desc

Returns

Return type None

template.runner.triplet package

Submodules

template.runner.triplet.evaluate module

template.runner.triplet.evaluate.**test**(*test_loader*, *model*, *writer*, *epoch*, *no_cuda=False*, *log_interval=20*, ***kwargs*)
Wrapper for _evaluate() with the intent to test the model

template.runner.triplet.evaluate.**validate**(*val_loader*, *model*, *writer*, *epoch*, *no_cuda=False*, *log_interval=20*, ***kwargs*)
Wrapper for _evaluate() with the intent to validate the model.

template.runner.triplet.setup module

template.runner.triplet.setup.**setup_dataloaders**(*model_expected_input_size*, *dataset_folder*, *n_triplets*, *batch_size*, *workers*, *inmem*, *only_evaluate=False*, ***kwargs*)
Set up the dataloaders for the specified datasets.

Parameters

- **model_expected_input_size** (*tuple*) – Specify the height and width that the model expects.
- **dataset_folder** (*string*) – Path string that points to the three folder train/val/test. Example: ~/../../data/svhn
- **n_triplets** (*int*) – Number of triplets to generate for train/val/tes
- **batch_size** (*int*) – Number of datapoints to process at once
- **workers** (*int*) – Number of workers to use for the dataloaders
- **inmem** (*boolean*) – Flag : if False, the dataset is loaded in an online fashion i.e. only file names are stored and images are loaded on demand. This is slower than storing everything in memory.
- **only_evaluate** (*boolean*) – Flag : if True, only the test set is loaded.

Returns

- **train_loader** (*torch.utils.data.DataLoader*)

- **val_loader** (*torch.utils.data.DataLoader*)
- **test_loader** (*torch.utils.data.DataLoader*) – Dataloaders for train, val and test.

template.runner.triplet.train module

template.runner.triplet.train.**train**(*train_loader*, *model*, *criterion*, *optimizer*, *writer*, *epoch*, *no_cuda*, *log_interval=25*, ***kwargs*)

Training routine

Parameters

- **train_loader** (*torch.utils.data.DataLoader*) – The dataloader of the train set.
- **model** (*torch.nn.module*) – The network model being used.
- **criterion** (*torch.nn.loss*) – The loss function used to compute the loss of the model.
- **optimizer** (*torch.optim*) – The optimizer used to perform the weight update.
- **writer** (*tensorboardX.writer.SummaryWriter*) – The tensorboard writer object. Used to log values on file for the tensorboard visualization.
- **epoch** (*int*) – Number of the epoch (for logging purposes).
- **no_cuda** (*boolean*) – Specifies whether the GPU should be used or not. A value of 'True' means the CPU will be used.
- **log_interval** (*int*) – Interval limiting the logging of mini-batches. Default value of 10.

Returns Placeholder 0. In the future this should become the FPR95

Return type int

template.runner.triplet.transforms module

class template.runner.triplet.transforms.**MultiCrop**(*size*, *n_crops*)

Bases: object

Crop the given PIL Image into multiple random crops

Parameters

- **size** (*tuple or int*) – Desired output size of the crop. If size is an int instead of sequence like (h, w), a square crop of size (size, size) is made.
- **n_crops** (*int*) – The number of crops to be generated from a page.

Returns

Return type None

Example

```
>>> MultiCrop(size=model_expected_input_size, n_crops=multi_crop),
>>> transforms.Lambda(lambda crops: torch.stack([transforms.ToTensor()(crop) for␣
↪crop in crops])),
>>> transforms.Lambda(lambda items: torch.stack([transforms.Normalize(mean=mean,␣
↪std=std)(item) for item in items]))
```

```
>>> transform = Compose([
>>> MultiCrop(size), # this is a list of PIL Images
>>> Lambda(lambda crops: torch.stack([ToTensor()(crop) for crop in crops])) #␣
↪returns a 4D tensor
>>> ])
>>> #In your test loop you can do the following:
>>> input, target = batch # input is a 5d tensor, target is 2d
>>> bs, ncrops, c, h, w = input.size()
>>> result = model(input.view(-1, c, h, w)) # fuse batch size and ncrops
>>> result_avg = result.view(bs, ncrops, -1).mean(1) # avg over crops
```

template.runner.triplet.transforms.**multi_crop**(*img*, *size*, *n_crops*)

Crop the given PIL Image into multiple random crops.

Parameters

- **img** (*PIL.Image*) – The Image to be processed.
- **size** (*tuple or int*) – Desired output size of the crop. If size is an `int` instead of sequence like (h, w), a square crop of size (size, size) is made.
- **n_crops** (*int*) – The number of crops to be generated from a page.

Returns **crops** – A list of PIL.Images which are the crops from the page.

Return type list of PIL.Images

template.runner.triplet.triplet module

This file is the template for the boilerplate of train/test of a triplet network. This code has initially been adapted to our purposes from http://www.iis.ee.ic.ac.uk/%7Evbalnt/shallow_descr/TFeat_paper.pdf

class template.runner.triplet.triplet.**Triplet**

Bases: `object`

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *margin*, *anchor_swap*, *validation_interval*, *regenerate_every*, *checkpoint_all_epochs*, *only_evaluate*, ***kwargs*)

This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **epochs** (*int*) – Number of epochs to train
- **lr** (*float*) – Value for learning rate

- **margin** (*float*) – The margin value for the triplet loss function
- **anchor_swap** (*boolean*) – Turns on anchor swap
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **regenerate_every** (*int*) – Re-generate triplets every N epochs
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.
- **only_evaluate** (*boolean*) – Flag : if True, only the test set is loaded.

Returns Mean Average Precision values for train and validation splits.

Return type train_value, val_value, test_value

Module contents

class template.runner.triplet.**Triplet**

Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *margin*, *anchor_swap*, *validation_interval*, *regenerate_every*, *checkpoint_all_epochs*, *only_evaluate*, ***kwargs*)

This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **epochs** (*int*) – Number of epochs to train
- **lr** (*float*) – Value for learning rate
- **margin** (*float*) – The margin value for the triplet loss function
- **anchor_swap** (*boolean*) – Turns on anchor swap
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **regenerate_every** (*int*) – Re-generate triplets every N epochs
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.
- **only_evaluate** (*boolean*) – Flag : if True, only the test set is loaded.

Returns Mean Average Precision values for train and validation splits.

Return type train_value, val_value, test_value

3.1.1.2 Module contents

class template.runner.**ApplyModel**
Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *lr*, *output_channels*, *classify*, ***kwargs*)
This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **lr** (*float*) – Value for learning rate
- **kwargs** (*dict*) – Any additional arguments.
- **output_channels** (*int*) – Specify shape of final layer of network.
- **classify** (*boolean*) – Specifies whether to generate a classification report for the data or not.

Returns **None** – None

Return type None

class template.runner.**Bidimensional**
Bases: *template.runner.image_classification.image_classification.ImageClassification*

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *validation_interval*, *checkpoint_all_epochs*, ***kwargs*)
This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard.SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **epochs** (*int*) – Number of epochs to train
- **lr** (*float*) – Value for learning rate
- **kwargs** (*dict*) – Any additional arguments.
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split
- **test_value** (*float*) – Accuracy value for test split

class template.runner.**DivahisdbSemanticSegmentation**
Bases: *template.runner.image_classification.image_classification.ImageClassification*

class_encoding = None

img_names_sizes_dict = None

classmethod prepare(*model_name*, ***kwargs*)
See parent class for documentation

class template.runner.**ImageClassification**
Bases: object

classmethod prepare(*model_name*, ***kwargs*)
Loads and prepares the data, the optimizer and the criterion

Parameters

- **model_name** (*str*) – Name of the model. Used for loading the model.
- **kwargs** (*dict*) – Any additional arguments.

Returns

- **model** (*DataParallel*) – The model to train
- **num_classes** (*int*) – How many different classes there are in our problem. Used for loading the model.
- **best_value** (*float*) – Best value of the model so far. Non-zero only in case of –resume being used
- **train_loader** (*torch.utils.data.dataloader.DataLoader*) – Training dataloader
- **val_loader** (*torch.utils.data.dataloader.DataLoader*) – Validation dataloader
- **test_loader** (*torch.utils.data.dataloader.DataLoader*) – Test set dataloader
- **optimizer** (*torch.optim*) – Optimizer to use during training, e.g. SGD
- **criterion** (*torch.nn.modules.loss*) – Loss function to use, e.g. cross-entropy

classmethod single_run(***kwargs*)
This is the main routine where train(), validate() and test() are called.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split
- **test_value** (*float*) – Accuracy value for test split

classmethod test_routine(*model_name*, *num_classes*, *criterion*, *epochs*, *current_log_folder*, *writer*, ***kwargs*)
Load the best model according to the validation score (early stopping) and runs the test routine.

Parameters

- **model_name** (*str*) – name of the model. Used for loading the model.
- **num_classes** (*int*) – How many different classes there are in our problem. Used for loading the model.
- **criterion** (*torch.nn.modules.loss*) – Loss function to use, e.g. cross-entropy
- **epochs** (*int*) – After how many epochs are we testing

- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **writer** (*Tensorboard.SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **kwargs** (*dict*) – Any additional arguments.

Returns **test_value** – Accuracy value for test split

Return type float

classmethod train_routine(*best_value*, *decay_lr*, *validation_interval*, *start_epoch*, *epochs*, *checkpoint_all_epochs*, *current_log_folder*, ***kwargs*)

Performs the training and validatation routines

Parameters

- **best_value** (*float*) – Best value of the model so far. Non-zero only in case of –resume being used
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **start_epoch** (*int*) – Int to initialize the starting epoch. Non-zero only in case of –resume being used
- **epochs** (*int*) – Number of epochs to train
- **checkpoint_all_epochs** (*bool*) – Save checkpoint at each epoch
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **kwargs** (*dict*) – Any additional arguments.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split

class template.runner.**MultiLabelImageClassification**

Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *validation_interval*, *checkpoint_all_epochs*, ***kwargs*)

This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard.SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **epochs** (*int*) – Number of epochs to train
- **lr** (*float*) – Value for learning rate
- **kwargs** (*dict*) – Any additional arguments.
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.

Returns

- **train_value** (ndarray[floats] of size (1, *epochs*)) – Accuracy values for train split
- **val_value** (ndarray[floats] of size (1, `epochs`+1)) – Accuracy values for validation split
- **test_value** (*float*) – Accuracy value for test split

class template.runner.**ProcessActivation**

Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *validation_interval*, *checkpoint_all_epochs*, ***kwargs*)

DESC

Parameters **Param** – Desc

Returns

Return type None

class template.runner.**SemanticSegmentation**

Bases: *template.runner.image_classification.image_classification.ImageClassification*

class_encoding = None

img_names_sizes_dict = None

classmethod prepare(*model_name*, ***kwargs*)

See parent class for documentation

class template.runner.**Triplet**

Bases: object

static single_run(*writer*, *current_log_folder*, *model_name*, *epochs*, *lr*, *decay_lr*, *margin*, *anchor_swap*, *validation_interval*, *regenerate_every*, *checkpoint_all_epochs*, *only_evaluate*, ***kwargs*)

This is the main routine where train(), validate() and test() are called.

Parameters

- **writer** (*Tensorboard SummaryWriter*) – Responsible for writing logs in Tensorboard compatible format.
- **current_log_folder** (*string*) – Path to where logs/checkpoints are saved
- **model_name** (*string*) – Name of the model
- **epochs** (*int*) – Number of epochs to train
- **lr** (*float*) – Value for learning rate
- **margin** (*float*) – The margin value for the triplet loss function
- **anchor_swap** (*boolean*) – Turns on anchor swap
- **decay_lr** (*boolean*) – Decay the lr flag
- **validation_interval** (*int*) – Run evaluation on validation set every N epochs
- **regenerate_every** (*int*) – Re-generate triplets every N epochs
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.
- **only_evaluate** (*boolean*) – Flag : if True, only the test set is loaded.

Returns Mean Average Precision values for train and validation splits.

Return type train_value, val_value, test_value

3.2 Submodules

3.3 template.CL_arguments module

template.CL_arguments.**parse_arguments**(*args=None*)
Argument Parser

3.4 template.RunMe module

This file is the main entry point of DeepDIVA.

We introduce DeepDIVA: an infrastructure designed to enable quick and intuitive setup of reproducible experiments with a large range of useful analysis functionality. Reproducing scientific results can be a frustrating experience, not only in document image analysis but in machine learning in general. Using DeepDIVA a researcher can either reproduce a given experiment or share their own experiments with others. Moreover, the framework offers a large range of functions, such as boilerplate code, keeping track of experiments, hyper-parameter optimization, and visualization of data and results.

It is completely open source and accessible as Web Service through DIVAService

>> Official website: https://diva-dia.github.io/DeepDIVAweb/ >> GitHub repo: https://github.com/DIVA-DIA/DeepDIVA >> Tutorials: https://diva-dia.github.io/DeepDIVAweb/articles.html

authors: Michele Alberti and Vinaychandran Pondenkandath (equal contribution)

class template.RunMe.**RunMe**
Bases: object

This class is used as entry point of DeepDIVA. The there are four main scenarios for using the framework:

- **Single run: (classic) run an experiment once with the given parameters specified by** command line. This is typical usage scenario.
- **Multi run: this will run multiple times an experiment. It basically runs the *single run*** scenario multiple times and aggregates the results. This is particularly useful to counter effects of randomness.
- **Optimize with SigOpt: this will start an hyper-parameter optimization search with the aid** of SigOpt (State-of-the-art Bayesian optimization tool). For more info on how to use it see the tutorial page on: https://diva-dia.github.io/DeepDIVAweb/articles.html
- **Optimize manually: this will start a grid-like hyper-parameter optimization with the** boundaries for the values specifies by the user in a provided file. This is much less efficient than using SigOpt but on the other hand is not using any commercial solutions.

main(*args=None*)
Select the use case based on the command line arguments and delegate the execution to the most appropriate sub-routine

Returns

- **train_scores** (ndarray[floats] of size (1, *epochs*) or None) – Score values for train split
- **val_scores** (ndarray[floats] of size (1, `epochs`+1) or None) – Score values for validation split

- **test_scores** (*float or None*) – Score value for test split

parser = None

3.5 template.setup module

template.setup.**copy_code**(*output_folder*)
Makes a tar file with DeepDIVA that exists during runtime.

Parameters **output_folder** (*str*) – Path to output directory

Returns

Return type None

template.setup.**set_up_dataloaders**(*model_expected_input_size*, *dataset_folder*, *batch_size*, *workers*, *disable_dataset_integrity*, *enable_deep_dataset_integrity*, *inmem=False*, ***kwargs*)
Set up the dataloaders for the specified datasets.

Parameters

- **model_expected_input_size** (*tuple*) – Specify the height and width that the model expects.
- **dataset_folder** (*string*) – Path string that points to the three folder train/val/test. Example: ~/../../data/svhn
- **batch_size** (*int*) – Number of datapoints to process at once
- **workers** (*int*) – Number of workers to use for the dataloaders
- **inmem** (*boolean*) – Flag: if False, the dataset is loaded in an online fashion i.e. only file names are stored and images are loaded on demand. This is slower than storing everything in memory.

Returns

- **train_loader** (*torch.utils.data.DataLoader*)
- **val_loader** (*torch.utils.data.DataLoader*)
- **test_loader** (*torch.utils.data.DataLoader*) – Dataloaders for train, val and test.
- *int* – Number of classes for the model.

template.setup.**set_up_env**(*gpu_id*, *seed*, *multi_run*, *no_cuda*, ***kwargs*)
Set up the execution environment.

Parameters

- **gpu_id** (*string*) – Specify the GPUs to be used
- **seed** (*int*) – Seed all possible seeds for deterministic run
- **multi_run** (*int*) – Number of runs over the same code to produce mean-variance graph.
- **no_cuda** (*bool*) – Specify whether to use the GPU or not

Returns

Return type None

template.setup.**set_up_logging**(*parser*, *experiment_name*, *output_folder*, *quiet*, *args_dict*, *debug*, ***kwargs*)
Set up a logger for the experiment

Parameters

- **parser** (*parser*) – The argument parser
- **experiment_name** (*string*) – Name of the experiment. If not specify, accepted from command line.
- **output_folder** (*string*) – Path to where all experiment logs are stored.
- **quiet** (*bool*) – Specify whether to print log to console or only to text file
- **debug** (*bool*) – Specify the logging level
- **args_dict** (*dict*) – Contains the entire argument dictionary specified via command line.

Returns

- **log_folder** (*String*) – The final logging folder tree
- **writer** (*tensorboardX.writer.SummaryWriter*) – The tensorboard writer object. Used to log values on file for the tensorboard visualization.

`template.setup.`**`set_up_model`**(*output_channels*, *model_name*, *pretrained*, *no_cuda*, *resume*, *load_model*, *disable_databalancing*, *dataset_folder*, *inmem*, *workers*, *optimizer_name=None*, *criterion_name=None*, *num_classes=None*, *ablate=False*, ***kwargs*)

Instantiate model, optimizer, criterion. Load a pretrained model or resume from a checkpoint.

Parameters

- **output_channels** (*int*) – Specify shape of final layer of network. Only used if num_classes is not specified.
- **model_name** (*string*) – Name of the model
- **pretrained** (*bool*) – Specify whether to load a pretrained model or not
- **optimizer_name** (*string*) – Name of the optimizer
- **criterion_name** (*string*) – Name of the criterion
- **no_cuda** (*bool*) – Specify whether to use the GPU or not
- **resume** (*string*) – Path to a saved checkpoint
- **load_model** (*string*) – Path to a saved model
- **start_epoch** (*int*) – Epoch from which to resume training. If if not resuming a previous experiment the value is 0
- **disable_databalancing** (*boolean*) – If True the criterion will not be fed with the class frequencies. Use with care.
- **dataset_folder** (*String*) – Location of the dataset on the file system
- **inmem** (*boolean*) – Load the whole dataset in memory. If False, only file names are stored and images are loaded on demand. This is slower than storing everything in memory.
- **workers** (*int*) – Number of workers to use for the dataloaders
- **num_classes** (*int*) – Number of classes for the model
- **ablate** (*boolean*) – If True, remove the final layer of the given model.

Returns

- **model** (*nn.Module*) – The actual model
- **criterion** (*nn.loss*) – The criterion for the network

- **optimizer** (*torch.optim*) – The optimizer for the model
- **best_value** (*float*) – Specifies the former best value obtained by the model. Relevant only if you are resuming training.

3.6 template.test_RunMe module

Warning: LONG RUNTIME TESTS!

This test suite is designed to verify that the main components of the framework are not broken. It is expected that smaller components or sub-parts are tested individually.

As we all know this will probably never happen, we will at least verify that the overall features are correct and fully functional. These tests will take long time to run and are not supposed to be run frequently. Nevertheless, it is important that before a PR or a push on the master branch the main functions can be tested.

Please keep the list of these tests up to date as soon as you add new features.

`template.test_RunMe.`**`test_one`**`()`

- Verify the sizes of the return of execute
- Image classification with default parameters

3.7 Module contents

CHAPTER

FOUR

UTIL PACKAGE

4.1 Subpackages

4.1.1 util.data package

4.1.1.1 Submodules

4.1.1.2 util.data.dataset_analytics module

This script perform some analysis on the dataset provided. In particular computes std and mean (to be used to center your dataset).

Structure of the dataset expected:

Split folders

'args.dataset-folder' has to point to the parent of the train folder. Example:

~/../../data/svhn

where the dataset_folder contains the train sub-folder as follow:

args.dataset_folder/train

Classes folders

The train split should have different classes in a separate folder with the class name. The file name can be arbitrary (e.g does not have to be 0-* for classes 0 of MNIST). Example:

train/dog/whatever.png train/dog/you.png train/dog/like.png

train/cat/123.png train/cat/nsdf3.png train/cat/asd932_.png

`util.data.dataset_analytics.`**`cms_inmem`**(*file_names*)

Computes mean and image_classification deviation in an offline fashion. This is possible only when the dataset can be allocated in memory.

Parameters **file_names** (*List of String*) – List of file names of the dataset

Returns

- **mean** (*double*)
- **std** (*double*)

util.data.dataset_analytics.**cms_online**(*file_names*, *workers*)
Computes mean and image_classification deviation in an online fashion. This is useful when the dataset is too big to be allocated in memory.

Parameters

- **file_names** (*List of String*) – List of file names of the dataset
- **workers** (*int*) – Number of workers to use for the mean/std computation

Returns

- **mean** (*double*)
- **std** (*double*)

util.data.dataset_analytics.**compute_mean_std**(*dataset_folder*, *inmem*, *workers*)
Computes mean and std of a dataset. Saves the results as CSV file in the dataset folder.

Parameters

- **dataset_folder** (*String (path)*) – Path to the dataset folder (see above for details)
- **inmem** (*Boolean*) – Specifies whether is should be computed i nan online of offline fashion.
- **workers** (*int*) – Number of workers to use for the mean/std computation

Returns

Return type None

util.data.dataset_analytics.**compute_mean_std_segmentation**(*dataset_folder*, *inmem*, *workers*, *filter_boundaries*)
Computes mean and std of a dataset for semantic segmentation. Saves the results as CSV file in the dataset folder.

Parameters

- **dataset_folder** (*String (path)*) – Path to the dataset folder (see above for details)
- **inmem** (*Boolean*) – Specifies whether is should be computed i nan online of offline fashion.
- **workers** (*int*) – Number of workers to use for the mean/std computation
- **filter_boundaries** (*bool*) – specifies whether thr boundary pixels should be removed or not

Returns

Return type None

4.1.1.3 util.data.dataset_bidimensional module

This script allows for creation of a bidimensional(2D) dataset.

util.data.dataset_bidimensional.**circle**(*size*)
Samples are generated in a grid fashion (np.linspace) and then draw a circle on x*x + y*y > 0.5 2 classes.

Parameters **size** (*int*) – The total number of points in the dataset.

Returns **train, val, test** – The three splits. Each row is (x,y,label)

Return type ndarray[float] of size (n,3)

util.data.dataset_bidimensional.**diagonal**(*size*)
Generates a dataset where points on a diagonal line are one class, and points surrounding it are a different class.

Parameters **size** (*int*) – The total number of points in the dataset.

Returns **train, val, test** – The three splits. Each row is (x,y,label)

Return type ndarray[float] of size (n,3)

util.data.dataset_bidimensional.**donut**(*size*)
Samples are generated in a grid fashion (np.linspace) and then draw a donut. 2 classes.

Parameters **size** (*int*) – The total number of points in the dataset.

Returns **train, val, test** – The three splits. Each row is (x,y,label)

Return type ndarray[float] of size (n,3)

util.data.dataset_bidimensional.**flag**(*size*)
XOR problem but with multi class, each corner a different class 4 classes.

Parameters **size** (*int*) – The total number of points in the dataset.

Returns **train, val, test** – The three splits. Each row is (x,y,label)

Return type ndarray[float] of size (n,3)

util.data.dataset_bidimensional.**spiral**(*size*)
Samples are generated in a two spiral fashion, starting from the center. 2 classes.

Parameters **size** (*int*) – The total number of points in the dataset.

Returns **train, val, test** – The three splits. Each row is (x,y,label)

Return type ndarray[float] of size (n,3)

util.data.dataset_bidimensional.**spiral_multi**(*size*)
Samples are generated in a two spiral fashion, starting from the center. 4 classes.

Parameters **size** (*int*) – The total number of points in the dataset.

Returns **train, val, test** – The three splits. Each row is (x,y,label)

Return type ndarray[float] of size (n,3)

util.data.dataset_bidimensional.**stripes**(*size*)
Samples are generated in a stripe fashion, like a TV color screen (vertical stripes). Each bin is a different class. 5 classes.

Parameters **size** (*int*) – The total number of points in the dataset.

Returns **train, val, test** – The three splits. Each row is (x,y,label)

Return type ndarray[float] of size (n,3)

util.data.dataset_bidimensional.**xor**(*size*)
XOR problem 2 classes.

Parameters **size** (*int*) – The total number of points in the dataset.

Returns **train, val, test** – The three splits. Each row is (x,y,label)

Return type ndarray[float] of size (n,3)

4.1.1.4 util.data.dataset_integrity module

This script generate the integrity footprint on the dataset provided. Such a footprint can be used to verify that the data has no been modified, altered or manipulated. The integrity of the dataset can be verified in two ways: quick and deep. The former is very fast and uses a high level type of verification such as recently modified files and file counts. The latter basically re-compute the footprint and verifies if it matches the existing one. This is slow and should be used only when the integrity of the dataset is a critical matter.

Structure of the dataset expected can be found at: https://diva-dia.github.io/DeepDIVAweb/articles/prepare-dataset/

`util.data.dataset_integrity.`**`dict_compare`**(*d1*, *d2*)

Parameters

- **d1** (*Dictionary*)
- **d2** (*Dictionary*) – Dictionaries to compare

Returns Sets with the element which has been respectively added, removed, modified or stayed the same

Return type added, removed, modified, same

`util.data.dataset_integrity.`**`generate_integrity_footprint`**(*dataset_folder*)

This function generates the integrity footprint on the dataset provided. Such a footprint can be used to verify that the data has no been modified, altered or manipulated.

The footprint file will contain the following information in a JSON format:

{ path : <string> // Path to this folder where the last step is the name of the folder last_modified : <date> // This correspond to the most recent 'last modified' in the dataset files : { // For each file

{ file_name : <string> // The filename as string file_hash : <hash> // This is the hash of the content

} folders : { // For each folder, recursion

// Recursion but NO last_modified (not needed anymore)

]

}

Parameters **dataset_folder** (*String (path)*) – Path to the dataset folder (see above for details)

Returns

Return type A dictionary of the format explained in generate_integrity_footprint() above.

`util.data.dataset_integrity.`**`get_last_modified`**(*dataset_folder*)

Elaborates the most recent 'last_modified' tag by scanning all files in the root folder and sub-folders.

This routine excludes the 'footprint.json' file which, if taken into account, would prevent the verification process to succeed (as it modifies the last modified of the root itself).

Parameters **dataset_folder** (*String (path)*) – Path to the dataset folder

Returns **last_modified** – A string representing the last modified of the entire folder

Return type String

`util.data.dataset_integrity.`**`save_footprint`**(*dataset_folder*, *filename*, *data*)

Save the footprint on file system

Parameters

- **dataset_folder** (*String (path)*) – Path to the dataset folder (see above for details)

- **filename** (*String*) – Name of the file where the data will be saved
- **data** (*dictionary*) – The actual data in JSON compliant format

Returns

Return type None

util.data.dataset_integrity.**verify_integrity_deep**(*dataset_folder*)
This function basically re-compute the footprint and verifies if it matches the existing one. This is slow and should be used only when the integrity of the dataset is a critical matter.

Parameters **dataset_folder** (*String (path)*) – Path to the dataset folder (see above for details)

Returns Is the dataset footprint still matching the data?

Return type Boolean

util.data.dataset_integrity.**verify_integrity_quick**(*dataset_folder*)
This function verifies that the 'last_modified' field still corresponds to the one contained in the footprint. This check is verify fast, but it comes at a price. The OS updates this number when files are added or removed to the folder, but NOT if a file is modified. Because of this, it is not 100% safe and especially does NOT protect you against malicious attacks! To have a safe check whether the data is the same you should rely on the slower verify_integrity_deep() function.

Parameters **dataset_folder** (*String (path)*) – Path to the dataset folder (see above for details)

Returns Is the 'last_modified' field still actual?

Return type Boolean

4.1.1.5 util.data.dataset_splitter module

This script allows for creation of a validation set from the training set.

util.data.dataset_splitter.**split_dataset**(*dataset_folder*, *split*, *symbolic*, *debug=False*)
Partition a dataset into train/val splits on the filesystem.

Parameters

- **dataset_folder** (*str*) – Path to the dataset folder (see datasets.image_folder_dataset.load_dataset for details).
- **split** (*float*) – Specifies how much of the training set should be converted into the validation set.
- **symbolic** (*bool*) – Does not make a copy of the data, but only symbolic links to the original data
- **debug** (*bool*) – Prints additional debug statements

Returns

Return type None

util.data.dataset_splitter.**split_dataset_segmentation**(*dataset_folder*, *split*, *symbolic*, *test=False*)
Partition a dataset into train/val(/test) splits on the filesystem for segmentation datasets organized as dataset/data with the images and dataset/gt for the ground truth. The corresponding images need to have the same name.

Parameters

- **dataset_folder** (*str*) – Path to the dataset folder (see datasets.image_folder_dataset.load_dataset for details).

- **split** (*float*) – Specifies how much of the training set should be converted into the validation set.
- **symbolic** (*bool*) – Does not make a copy of the data, but only symbolic links to the original data
- **test** (*bool*) – If true, the validation set is split again (1:1) into a val and test set. Default false.

Returns

Return type None

util.data.dataset_splitter.**split_dataset_writerIdentification**(*dataset_folder*, *split*)

Partition a dataset into train/val splits on the filesystem.

Parameters

- **dataset_folder** (*str*) – Path to the dataset folder (see datasets.image_folder_dataset.load_dataset for details).
- **split** (*float*) – Specifies how much of the training set should be converted into the validation set.
- **symbolic** (*bool*) – Does not make a copy of the data, but only symbolic links to the original data

Returns

Return type None

4.1.1.6 util.data.get_a_dataset module

util.data.get_a_dataset.**cifar10**(*args*)

Fetches and prepares (in a DeepDIVA friendly format) the CIFAR dataset to the location specified on the file system

Parameters **args** (*dict*) – List of arguments necessary to run this routine. In particular its necessary to provide output_folder as String containing the path where the dataset will be downloaded

Returns

Return type None

util.data.get_a_dataset.**diva_hisdb**(*args*)

Fetches and prepares (in a DeepDIVA friendly format) the DIVA HisDB-all dataset for semantic segmentation to the location specified on the file system

See also: https://diuf.unifr.ch/main/hisdoc/diva-hisdb

Output folder structure: ../HisDB/CB55/train ../HisDB/CB55/val ../HisDB/CB55/test

../HisDB/CB55/test/data -> images ../HisDB/CB55/test/gt -> pixel-wise annotated ground truth

Parameters **args** (*dict*) – List of arguments necessary to run this routine. In particular its necessary to provide output_folder as String containing the path where the dataset will be downloaded

Returns

Return type None

util.data.get_a_dataset.**fashion_mnist**(*args*)
Fetches and prepares (in a DeepDIVA friendly format) the Fashion-MNIST dataset to the location specified on the file system

Parameters **args** (*dict*) – List of arguments necessary to run this routine. In particular its necessary to provide output_folder as String containing the path where the dataset will be downloaded

Returns

Return type None

util.data.get_a_dataset.**glas**(*args*)
Fetches and prepares (in a DeepDIVA friendly format) the tubule dataset (from the GlaS challenge) for semantic segmentation to the location specified on the file system

See also: https://github.com/choosehappy/public/tree/master/DL%20tutorial%20Code/3-tubule

Output folder structure: ../HisDB/GlaS/train ../HisDB/GlaS/val ../HisDB/GlaS/test

../HisDB/GlaS/test/data -> images ../HisDB/GlaS/test/gt -> pixel-wise annotated ground truth

Parameters **args** (*dict*) – List of arguments necessary to run this routine. In particular its necessary to provide output_folder as String containing the path where the dataset will be downloaded

Returns

Return type None

util.data.get_a_dataset.**historical_wi**(*args*)

util.data.get_a_dataset.**icdar2017_clamm**(*args*)

util.data.get_a_dataset.**kmnist**(*args*)
Fetches and prepares (in a DeepDIVA friendly format) the K-MNIST dataset to the location specified on the file system

Parameters **args** (*dict*) – List of arguments necessary to run this routine. In particular its necessary to provide output_folder as String containing the path where the dataset will be downloaded

Returns

Return type None

util.data.get_a_dataset.**miml**(*args*)
Fetches and prepares (in a DeepDIVA friendly format) the Multi-Instance Multi-Label Image Dataset on the file system. Dataset available at: http://lamda.nju.edu.cn/data_MIMLimage.ashx

Parameters **args** (*dict*) – List of arguments necessary to run this routine. In particular its necessary to provide output_folder as String containing the path where the dataset will be downloaded

Returns

Return type None

util.data.get_a_dataset.**mnist**(*args*)
Fetches and prepares (in a DeepDIVA friendly format) the MNIST dataset to the location specified on the file system

Parameters **args** (*dict*) – List of arguments necessary to run this routine. In particular its necessary to provide output_folder as String containing the path where the dataset will be downloaded

Returns

Return type None

util.data.get_a_dataset.**svhn**(*args*)
Fetches and prepares (in a DeepDIVA friendly format) the SVHN dataset to the location specified on the file system

Parameters **args** (*dict*) – List of arguments necessary to run this routine. In particular its necessary to provide output_folder as String containing the path where the dataset will be downloaded

Returns

Return type None

4.1.1.7 util.data.remove_whitespace module

util.data.remove_whitespace.**get_list_images**(*dir*)

util.data.remove_whitespace.**main**(*args*)

util.data.remove_whitespace.**open_crop_save**(*path*)

util.data.remove_whitespace.**remove_empty**(*img*)

4.1.1.8 util.data.shuffle_labels module

This script allows creates a symlink directory with all labels shuffled.

util.data.shuffle_labels.**split_dataset**(*dataset_folder*, *output_folder*, *symbolic*)
Partition a dataset into train/val splits on the filesystem.

Parameters

- **dataset_folder** (*str*) – Path to the dataset folder (see datasets.image_folder_dataset.load_dataset for details).
- **output_folder** (*str*) – Path to the output folder (see datasets.image_folder_dataset.load_dataset for details).
- **symbolic** (*bool*) – Does not make a copy of the data, but only symbolic links to the original data

Returns

Return type None

4.1.1.9 Module contents

4.1.2 util.evaluation package

4.1.2.1 Subpackages

util.evaluation.metrics package

Submodules

util.evaluation.metrics.accuracy module

util.evaluation.metrics.accuracy.**accuracy**(*predicted*, *target*, *topk=1*)
Computes the accuracy@K for the specified values of K

From https://github.com/pytorch/examples/blob/master/imagenet/main.py

Parameters

- **predicted** (*torch.FloatTensor*) – The predicted output values of the model. The size is batch_size x num_classes
- **target** (*torch.LongTensor*) – The ground truth for the corresponding output. The size is batch_size x 1
- **topk** (*tuple*) – Multiple values for K can be specified in a tuple, and the different accuracies@K will be computed.

Returns **res** – List of accuracies computed at the different K's specified in *topk*

Return type list

`util.evaluation.metrics.accuracy.`**`accuracy_segmentation`**(*label_trues*, *label_preds*, *n_class*)

Taken from https://github.com/wkentaro/pytorch-fcn Calculates the accuracy measures for the segmentation runner

Parameters

- **label_trues** (*matrix (batch size x H x W)*) – contains the true class labels for each pixel
- **label_preds** (*matrix ((batch size x H x W)*) – contains the predicted class for each pixel
- **n_class** (*int*) – number possible classes
- **border_pixel** (*boolean*) – true if border pixel value should be

Returns

Return type overall accuracy, mean accuracy, mean IU, fwavacc

util.evaluation.metrics.apk module

`util.evaluation.metrics.apk.`**`apk`**(*query*, *predicted*, *k='full'*)

Computes the average precision@k.

Parameters

- **query** (*int*) – Query label.
- **predicted** (*List(int)*) – Ordered list where each element is a label.
- **k** (*str or int*) – If int, cutoff for retrieval is set to K If str, 'full' means cutoff is til the end of predicted

  'auto' means cutoff is set to number of relevant queries.

  Example: query = 0 predicted = [0, 0, 1, 1, 0] if k == 'full', then k is set to 5 if k == 'auto', then k is set to num of 'query' values in 'predicted', i.e., k=3 as there as 3 of them in 'predicted'

Returns Average Precision@k

Return type float

`util.evaluation.metrics.apk.`**`compute_mapk`**(*distances*, *labels*, *k*, *workers=None*)

Convenience function to convert a grid of pairwise distances to predicted elements, to evaluate mean average precision (at K).

Parameters

- **distances** (*ndarray*) – A numpy array containing pairwise distances between all elements
- **labels** (*list*) – Ground truth labels for every element
- **k** (*int*) – Maximum number of predicted elements

Returns

- *float* – The mean average precision@K.
- *dict{label, float}* – The per class mean averages precision @k

util.evaluation.metrics.apk.**mapk**(*query*, *predicted*, *k=None*, *workers=1*)

Compute the mean Average Precision@K.

Parameters

- **query** (*list*) – List of queries.
- **predicted** (*list of list, or generator to list of lists*) – Predicted responses for each query. Supports chunking with slices in the first dimension.
- **k** (*str or int*) – If int, cutoff for retrieval is set to *k* If str, 'full' means cutoff is til the end of predicted

  'auto' means cutoff is set to number of relevant queries.

  For e.g., *query* = 0 *predicted* = [0, 0, 1, 1, 0] if *k* == 'full', then *k* is set to 5 if *k* == 'auto', then *k* is set to num of *query* values in *predicted*, i.e., *k`=3 as there as 3 of them in `predicted*.

- **workers** (*int*) – Number of parallel workers used to compute the AP@k

Returns

- *float* – The mean average precision@K.
- *dict{label, float}* – The per class mean averages precision @k

util.evaluation.metrics.test_accuracy module

util.evaluation.metrics.test_accuracy.**test_mini_batch**()

util.evaluation.metrics.test_accuracy.**test_no_batch**()

util.evaluation.metrics.test_apk module

Module contents

4.1.2.2 Submodules

4.1.2.3 util.evaluation.similarity_sort_embedding module

4.1.2.4 Module contents

4.1.3 util.visualization package

4.1.3.1 Submodules

4.1.3.2 util.visualization.confusion_matrix_heatmap module

util.visualization.confusion_matrix_heatmap.**make_heatmap**(*confusion_matrix*, *class_names*)

This function prints and plots the confusion matrix.

Adapted from https://gist.github.com/shaypal5/94c53d765083101efc0240d776a23823

Parameters

- **confusion_matrix** (*numpy.ndarray*) – Array containing the confusion matrix to be plotted
- **class_names** (*list of strings*) – Names of the different classes

Returns **data** – Contains an RGB image of the plotted confusion matrix

Return type numpy.ndarray

4.1.3.3 util.visualization.decision_boundaries module

util.visualization.decision_boundaries.**plot_decision_boundaries**(*output_winners*, *output_confidence*, *grid_x*, *grid_y*, *point_x*, *point_y*, *point_class*, *num_classes*, *step*, *writer*, *epochs*, ***kwargs*)

Plots the decision boundaries as a 2D image onto Tensorboard.

Parameters

- **output_winners** (*numpy.ndarray*) – which class is the 'winner' of the network at each location
- **output_confidence** (*numpy.ndarray*) – confidence value of the network for the 'winner' class
- **grid_x** (*numpy.ndarray*) – X axis locations of the decision grid
- **grid_y** (*numpy.ndarray*) – Y axis locations of the decision grid
- **point_x** (*numpy.ndarray*) – X axis locations of the real points to be plotted

- **point_y** (*numpy.ndarray*) – Y axis locations of the real points to be plotted
- **point_class** (*numpy.ndarray*) – class of the real points at each location
- **writer** (*tensorboardX SummaryWriter*) – Tensorboard summarywriter object
- **num_classes** (*int*) – number of unique classes
- **step** (*int*) – global training step
- **epochs** (*int*) – total number of training epochs

Returns

Return type None

4.1.3.4 util.visualization.embedding module

This script generates embedding visualization for features produced by the apply_model script.

`util.visualization.embedding.`**`isomap`**(*features*, *n_components=2*)

Returns the embedded points for Isomap. :Parameters: * **features** (*numpy.ndarray*) – contains the input feature vectors.

- **n_components** (*int*) – number of components to transform the features into

Returns **embedding** – x,y(z) points that the feature vectors have been transformed into

Return type numpy.ndarray

`util.visualization.embedding.`**`mds`**(*features*, *n_components=2*)

Returns the embedded points for MDS. :Parameters: * **features** (*numpy.ndarray*) – contains the input feature vectors.

- **n_components** (*int*) – number of components to transform the features into

Returns **embedding** – x,y(z) points that the feature vectors have been transformed into

Return type numpy.ndarray

`util.visualization.embedding.`**`pca`**(*features*, *n_components=2*)

Returns the embedded points for PCA. :Parameters: * **features** (*numpy.ndarray*) – contains the input feature vectors.

- **n_components** (*int*) – number of components to transform the features into

Returns **embedding** – x,y(z) points that the feature vectors have been transformed into

Return type numpy.ndarray

`util.visualization.embedding.`**`tsne`**(*features*, *n_components=2*)

Returns the embedded points for TSNE. :Parameters: * **features** (*numpy.ndarray*) – contains the input feature vectors.

- **n_components** (*int*) – number of components to transform the features into

Returns **embedding** – x,y(z) points that the feature vectors have been transformed into

Return type numpy.ndarray

4.1.3.5 util.visualization.mean_std_plot module

util.visualization.mean_std_plot.**plot_mean_std**(*x=None*, *arr=None*, *suptitle=''*, *title=''*, *xlabel='X'*, *ylabel='Y'*, *xlim=None*, *ylim=None*)

Plots the accuracy/loss curve over several runs with standard deviation and mean. :Parameters: * **x** (*numpy.ndarray*) – contains the ticks on the x-axis

- **arr** (*numpy.ndarray*) – contains the accuracy values for each epoch per run
- **suptitle** (*str*) – title for the plot
- **title** (*str*) – sub-title for the plot
- **xlabel** (*str*) – label for the x-axis
- **ylabel** (*str*) – label for the y-axis
- **xlim** (*float or None*) – optionally specify a upper limit on the x-axis
- **ylim** (*float or None*) – optionally specify a upper limit on the y-axis

Returns **data** – Contains an RGB image of the plotted accuracy curves

Return type numpy.ndarray

4.1.3.6 util.visualization.visualize_activations module

This script generates visualizations of the activation of intermediate layers of CNNs.

util.visualization.visualize_activations.**main**(*args*)

Main routine of script to generate activation heatmaps. :Parameters: **args** (*argparse.Namespace*) – contains all arguments parsed from input

Returns

Return type None

util.visualization.visualize_activations.**make_grid_own**(*activations*)

Plots all activations of a layer in a grid format. :Parameters: **activations** (*numpy.ndarray*) – array of activation values for each filter in a layer

Returns **large_fig** – image array containing all activation heatmaps of a layer

Return type numpy.ndarray

4.1.3.7 Module contents

4.2 Submodules

4.3 util.misc module

General purpose utility functions.

class util.misc.**AverageMeter**

Bases: object

Computes and stores the average and current value

From https://github.com/pytorch/examples/blob/master/imagenet/main.py

reset()

update(*val*, *n=1*)

util.misc.**HSVToRGB**(*h*, *s*, *v*)

util.misc.**adjust_learning_rate**(*lr*, *optimizer*, *epoch*, *decay_lr_epochs*, ***kwargs*)
Sets the learning rate to the initial LR decayed by 10 every N epochs.

Adapted from https://github.com/pytorch/examples/blob/master/imagenet/main.py

Parameters

- **lr** (*float*) – Learning rate.
- **optimizer** (*torch.optim object*) – The optimizer used for training the network.
- **epoch** (*int*) – Current training epoch.
- **decay_lr_epochs** (*int*) – Change the learning rate every N epochs.

Returns

Return type None

util.misc.**checkpoint**(*epoch*, *new_value*, *best_value*, *model*, *optimizer*, *log_dir*, *invert_best=False*, *checkpoint_all_epochs=False*, ***kwargs*)
Saves the current training checkpoint and the best valued checkpoint to file.

Parameters

- **epoch** (*int*) – Current epoch, for logging purpose only.
- **new_value** (*float*) – Current value achieved by the model at this epoch. To be compared with 'best_value'.
- **best_value** (*float*) – Best value ever obtained (so the last checkpointed model). To be compared with 'new_value'.
- **model** (*torch.nn.module object*) – The model we are checkpointing, this can be saved on file if necessary.
- **optimizer** – The optimizer that is being used to train this model. It is necessary if we were to resume the training from a checkpoint.
- **log_dir** (*str*) – Output folder where to put the model.
- **invert_best** (*bool*) – Changes the scale such that smaller values are better than bigger values (useful when metric evaluted is error rate)
- **checkpoint_all_epochs** (*bool*) – If enabled, save checkpoint after every epoch.
- **kwargs** (*dict*) – Any additional arguments.

Returns **best_value** – Best value ever obtained.

Return type float

util.misc.**get_all_files_in_folders_and_subfolders**(*root_dir=None*)
Get all the files in a folder and sub-folders.

Parameters **root_dir** (*str*) – All files in this directory and it's sub-folders will be returned by this method.

Returns **paths** – List of paths to all files in this folder and it's subfolders.

Return type list of str

util.misc.**get_distinct_colors**(*n*)

util.misc.**has_extension**(*filename*, *extensions*)
Checks if a file is an allowed extension.

Adapted from https://github.com/pytorch/vision/blob/master/torchvision/datasets/folder.py.

Parameters

- **filename** (*string*) – path to a file
- **extensions** (*list*) – extensions to match against

Returns True if the filename ends with one of given extensions, false otherwise.

Return type bool

util.misc.**int_to_one_hot**(*x*, *n_classes*)
Read out class encoding from blue channel bit-encoding (1 to [0,0,0,1] -> length determined by the number of classes)

Parameters

- **x** (*int*) – (pixel value of Blue channel from RGB image)
- **n_classes** (*int*) – number of class labels

Returns (multi) one-hot encoded list for integer

Return type list

util.misc.**load_numpy_image**(*dest_filename*)

util.misc.**make_colour_legend_image**(*img_name*, *colour_encoding*)

util.misc.**make_folder_if_not_exists**(*path*)

util.misc.**multi_label_img_to_multi_hot**(*np_array*)
TODO: There must be a faster way of doing this + ajust to correct input format (see gt_tensor_to_one_hot)
Convert ground truth label image to multi-one-hot encoded matrix of size image height x image width x #classes

Parameters **np_array** (*numpy array*) – RGB image [W x H x C]

Returns sparse one-hot encoded multi-class matrix, where #C is the number of classes

Return type numpy array of size [#C x W x H]

util.misc.**multi_one_hot_to_output**(*matrix*)
This function converts the multi-one-hot encoded matrix to an image like it was provided in the ground truth

Parameters **tensor of size [#C x W x H]** – sparse one-hot encoded multi-class matrix, where #C is the number of classes

Returns **np_array** – RGB image [C x W x H]

Return type numpy array

util.misc.**pil_loader**(*path*)

util.misc.**save_image_and_log_to_tensorboard**(*writer=None*, *tag=None*, *image=None*, *global_step=None*)
Utility function to save image in the output folder and also log it to Tensorboard.

Parameters

- **writer** (*tensorboardX.writer.SummaryWriter object*) – The writer object for Tensorboard
- **tag** (*str*) – Name of the image.

- **image** (*ndarray [W x H x C]*) – Image to be saved and logged to Tensorboard.
- **global_step** (*int*) – Epoch/Mini-batch counter.

Returns

Return type None

util.misc.**save_numpy_image**(*dest_filename*, *image*)

util.misc.**tensor_to_image**(*image*)
Tries to reshape, convert and do operations necessary to bring the image in a format friendly to be saved and logged to Tensorboard by save_image_and_log_to_tensorboard()

Parameters **image** (*?*) – Image to be converted

Returns **image** – Image, as format friendly to be saved and logged to Tensorboard.

Return type ndarray [W x H x C]

util.misc.**to_capital_camel_case**(*s*)
Converts a string to camel case.

Parameters **s** (*str*) – Input string.

Returns Input string *s* converted to camel case.

Return type str

4.4 Module contents

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

m

t

u

INDEX

O

P

R

S

T

U

V

X