Capsule Network for Road Sign Classification

Release 0.2

Aliaksandr Nekrashevich

Nov 27, 2023

CONTENTS:

CHAPTER

ONE

CAPSNET

1.1 capsnet package

1.1.1 Submodules

1.1.2 capsnet.benchmark module

Implements Benchmarks and related API. Benchmark is designed to process all job related to loading and transforming images.

`class capsnet.benchmark.`**`BelgiumTraffic`**(*config: ConfigBenchmark*)

Bases: *`IBenchmark`*

Belgium Traffic Benchmark.

`__init__`(*config: ConfigBenchmark*) → None

Initialization method. Sets datasets and initializes loaders.

Parameters

config – Benchmark configuration.

`class capsnet.benchmark.`**`ChineseTraffic`**(*config: ConfigBenchmark*)

Bases: *`IBenchmark`*

Chinese Traffic Benchmark.

`__init__`(*config: ConfigBenchmark*) → None

Initialization method. Sets datasets and initializes loaders.

Parameters

config – Benchmark configuration.

`class capsnet.benchmark.`**`GermanTraffic`**(*config: ConfigBenchmark*)

Bases: *`IBenchmark`*

German Traffic Benchmark.

`__init__`(*config: ConfigBenchmark*) → None

Initialization method. Sets datasets and initializes loaders.

Parameters

config – Benchmark configuration.

`class capsnet.benchmark.IBenchmark`(*config: ConfigBenchmark*)

Bases: `object`

Benchmark Interface. Benchmark is designed to process all job related to loading and transforming images.

Variables

- **normalize_transform** – Normalization transformation. Standard scaling for images.
- **recover_normalize** – Inverse transformatino to normalize_transform.
- **static_transform** – Static transform version. Used at evaluation.
- **random_transform** – Random transform version. Used at training.
- **train_dataset** – The train dataset, benchmark-specific DynamicDataset for training part.
- **test_dataset** – The test dataset, benchmark-specific DynamicDataset for testing part.
- **train_loader** – Loader for training part.
- **test_loader** – Loader for testing part.
- **use_cuda** – Whether to use CUDA.
- **num_workers** – Number of parallel workers in loaders.

`__init__`(*config: ConfigBenchmark*) → None

Initializer method. Prepares static and random transformations.

Parameters

config – Benchmark configuration.

`init_loaders`(*config: ConfigBenchmark*) → None

Loader initialization. Sets train dataset to random mode, test dataset to static mode, and calls loader-specific methods.

Parameters

config – Benchmark configuration.

`reset_test_loader`(*batch_size: int*, *shuffle_switch: bool*) → None

Resets/Initializes test data loader.

Parameters

- **batch_size** – The size of batch.
- **shuffle_switch** – Whether to shuffle the data.

`reset_train_loader`(*batch_size: int*) → None

Resets/Initializes train data loader.

Parameters

batch_size – The size of batch.

`set_random_mode()` → None

Sets train dataset to random transformation mode.

`set_static_mode()` → None

Sets train dataset to static transformation mode.

`class capsnet.benchmark.RussianTraffic`(*config: ConfigBenchmark*)

Bases: *IBenchmark*

Russian Traffic Benchmark.

__init__(*config: ConfigBenchmark*) → None

Initialization method. Sets datasets and initializes loaders.

Parameters
config – Benchmark configuration.

capsnet.benchmark.**build_benchmark**(*config: ConfigBenchmark*) → *IBenchmark*

Benchmark builder method. Creates corresponding benchmark according to benchmark configuration. If needed, estimates normalization parameters before creating benchmark.

Parameters
config – Benchmark configuration.

Returns
Ready-to-use benchmark supporting IBenchmark interface.

1.1.3 capsnet.checkpoint module

Implements API for saving and loading checkpoints.

capsnet.checkpoint.**load_checkpoint**(*checkpoint_path: str*, *model: Module*, *optimizer: Optimizer*, *use_cuda: bool*) → int

Loads model parameters from checkpoint.

Parameters

- **checkpoint_path** – The path to checkpoint.
- **model** – The network module for loading model parameters.
- **optimizer** – The optimizer module for loading optimizer parameters.
- **use_cuda** – Whether to use CUDA.

Returns
The index of the epoch corresponding to loaded checkpoint.

capsnet.checkpoint.**save_checkpoint**(*checkpoint_path: str*, *epoch: int*, *model: Module*, *optimizer: Optimizer*, *test_loss: float*, *test_accuracy: float*, *train_loss: float*, *train_accuracy: float*) → None

Saves current model state into a checkpoint.

Parameters

- **checkpoint_path** – The path to save checkpoint.
- **epoch** – The number of current epoch.
- **model** – The current model. For saving, state_dict is extracted.
- **optimizer** – The current optimizer. For saving, state_dict is extracted.
- **test_loss** – The value of current test loss.
- **test_accuracy** – The value of current test accuracy.
- **train_loss** – The value of current train loss.
- **train_accuracy** – The value of current train accuracy.

1.1.4 capsnet.config module

Specifies configs for entire experiment and provides default values.

`class capsnet.config.ConfigAgreement`

Bases: `object`

Configures agreement routing.

Variables

- **num_input_caps** – The number of input capsules.
- **num_output_caps** – The number of output capsules.
- **n_iterations** – The number of routing cycles.
- **output_caps_dim** – The dimension of output capsules.
- **use_cuda** – Whether to use CUDA.

`__init__()` → None

Initializer method.

`class capsnet.config.ConfigBenchmark`

Bases: `object`

Configures benchmarks.

Variables

- **num_load_workers** – Number of loading workers for DataLoader.
- **benchmark** – The name of the benchmark.
- **image_size** – The size of the image.
- **image_color** – The image color model.
- **num_channels** – The number of image channels.
- **use_augmentation** – Whether to use augmentation.
- **augment_proba** – Probability to augment.
- **random_entry_proba** – Probability to apply each transformation in the sequence.
- **estimate_normalization** – Whether to estimate normalization.
- **n_point_to_estimate** – Number of points to use for normalization estimation.
- **mean_normalize** – Mean value for image normalization.
- **std_normalize** – Standard deviation for image normalization.
- **batch_size** – The batch size.
- **use_cuda** – Whether to use CUDA.

`__init__()` → None

Initializer method.

`class capsnet.config.ConfigConv`

Bases: `object`

Configures convolution module.

Variables

- **in_channels** – The number of input channels.
- **out_channels** – The number of output channels.
- **kernel_size** – The kernel size.
- **use_batch_norm** – Whether to use batch normalization.
- **stride** – The stride value.

__init__() → None

Initializer method.

class capsnet.config.**ConfigNetwork**

Bases: object

Configures Capsule Network.

Variables

- **conv_config** – Configuration for initial convolution layer.
- **primary_config** – Configuration for primary capsules.
- **squash_config** – Configuration for squashing module.
- **agreement_config** – Configuration for agreement routing.
- **recognition_config** – Configuration for recognition capsules.
- **reconstruction_config** – Configuration for reconstruction network.
- **net_reconstruction_loss_reg** – Regularizer coefficient for reconstruction loss.
- **net_margin_loss_blend** – Blending coefficient inside margin loss.
- **net_margin_upper** – Upper margin value.
- **net_margin_lower** – Lower maring value.
- **use_square_in_margin_loss** – Whether to use squares in margin loss.

__init__() → None

Initializer method.

class capsnet.config.**ConfigPrimary**

Bases: object

Configures primary capsules.

Variables

- **num_capsules** – The number of primary capsules.
- **in_conv_channels** – The number of input convolution channels.
- **out_conv_channels** – The number of output convolution channels.
- **conv_kernel_size** – The size of convolution kernel.
- **conv_stride** – The stride of convolution kernel.
- **conv_padding** – The convolution padding.
- **capsule_output_dim** – The output dimension of each capsule.
- **use_dropout** – Whether to use dropout.
- **dropout_proba** – Dropout probability.

- **use_nan_gradient_hook** – Whether to use the NaN gradient hook.

__init__() → None

Initializer method.

class capsnet.config.**ConfigRecognition**

Bases: object

Configures recognition capsules.

Variables

- **num_output_caps** – The number of the output capsules.
- **output_caps_dim** – The dimensionality of the output capsules.
- **num_input_caps** – The number of input capsules.
- **input_caps_dim** – The dimensionality of the input capsules.
- **use_dropout** – Whether to use dropout.
- **dropout_proba** – The probability of dropout.
- **use_nan_gradient_hook** – Whether to use NaN gradient hook.

__init__() → None

Initializer method.

class capsnet.config.**ConfigReconstruction**

Bases: object

Configures reconstruction network.

Variables

- **linear_input_dim** – Input dimensionality.
- **linear_hidden_layers** – List with sizes of hidden layers.
- **linear_ouptut_dim** – Output dimensionality.
- **num_classes** – The number of classes.
- **use_cuda** – Whether to use CUDA.
- **output_img_size** – The size of ouptut image.
- **output_n_channels** – The number of output channels.

__init__() → None

Initializer method.

class capsnet.config.**ConfigSquash**

Bases: object

Configures squashing module.

Variables

- **eps_denom** – Shift for denominator for numeric stability.
- **eps_sqrt** – Shift for square root for numeric stability.
- **eps_input** – Shift for input for numeric stability.
- **eps_norm** – Shift for norm for numeric stability.

`__init__()` → None

Initializer method.

`class capsnet.config.ConfigTraining`

Bases: `object`

Configures network training process.

Parameters

- **debug_mode** – Whether the launch is for debugging. Debug mode makes process slower but outputs more in-process information.
- **load_checkpoint** – Whether to load checkpoint.
- **path_to_checkpoint** – Path to checkpoint for loading.
- **dump_checkpoints** – Whether to dump checkpoints during training process.
- **checkpoint_root** – Path to checkpoint root.
- **checkpoint_template** – The template for saving checkpoint files.
- **use_cuda** – Whether to use CUDA.
- **n_epochs** – Number of training epochs.
- **batch_size** – The size of training batch.
- **n_classes** – Number of classes.
- **use_clipping** – Whether to use gradient clipping.
- **clipping_threshold** – Gradient clipping threshold.
- **log_frequency** – The frequency of logging.
- **checkpoint_frequency** – The frequency of creating checkpoints.
- **n_visualize** – The number of reconstructed images to visualize.
- **graph_to_tensorboard** – Whether to save graph to TensorBoard.
- **use_lime** – Whether to use LIME and provide explanations.

`__init__()` → None

Initializer method.

`class capsnet.config.SetupConfig`

Bases: `object`

Configures the entire experiment.

Variables

- `benchmark_config` – The benchmark configuration.
- `network_config` – The capsule network configuration.
- `training_config` – The training process configuration.

`__init__()` → None

Initializer method.

1.1.5 capsnet.dataset module

Implements Dataset API - family of PyTorch-friendly benchmark-specific classes used for transforming image input and creating batches.

`class capsnet.dataset.BelgiumDataset`(*path_to_img_dir: str*, *path_to_annotations: Optional[str]*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*)

Bases: `DynamicDataset`

Specifies DynamicDataset for Belgium benchmark.

`__init__`(*path_to_img_dir: str*, *path_to_annotations: Optional[str]*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*) → None

Initializer method.

Parameters

- **path_to_img_dir** – Path to root directory with dataset images.
- **path_to_annotations_file** – Path to file with annotations. Irrelevant for this dataset, since class labels are encoded in file names.
- **static_transform** – Static image transformation version (used at evaluation).
- **random_transform** – Dynamic image transformation version (used at training).

`class capsnet.dataset.ChineseDataset`(*path_to_img_dir: str*, *path_to_annotations: str*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*)

Bases: `DynamicDataset`

Specifies DynamicDataset for Chinese benchmark.

Variables

`annotation_df` – The Pandas DataFrame with annotations.

`__init__`(*path_to_img_dir: str*, *path_to_annotations: str*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*) → None

Initializer method.

Parameters

- **path_to_img_dir** – Path to root directory with dataset images.
- **path_to_annotations_file** – Path to file with annotations.
- **static_transform** – Static image transformation version (used at evaluation).
- **random_transform** – Dynamic image transformation version (used at training).

`class capsnet.dataset.DynamicDataset`(*root_dir: str*, *annotations_path: Optional[str]*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*)

Bases: `Dataset`, `ABC`

DynamicDataset is a PyTorch friendly proxy for reading data and creating image batches. The parent class Dataset does the vast majority of the job, so this class only implements some methods to fill the gaps between pre-implemented and modified API.

Variables

- `static_transform` – Static initial transformation version, used at evaluation.

- `random_transform` – Random initial transformation version, used at training.
- `transform` – The currently set initial tranformation.
- `idx2class` – Maps entry index to class.
- `idx2name` – Maps entry index to relative path (often simply image name).
- `root_dir` – Path to root directory with dataset images.
- `annotations_path` – Path to file with annotations, when relevant.
- `n_classes` – Number of classes.

`__getitem__`(*idx: Union[Tensor, int]*) → Tuple[Image, int]

Getter by index. Method needed re-implementation to link prepared and desired APIs.

Parameters
idx – Index of the object to get.

Returns

Tuple with two elements. The first element is the transformed
image, the second element is the image class label.

`__init__`(*root_dir: str*, *annotations_path: Optional[str]*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*) → None

Initializer method.

Parameters

- **root_dir** – Path to root directory with dataset images.
- **annotations_path** – Path to annotation file, if relevant.
- **static_transform** – Static image transformation version (used at evaluation).
- **random_transform** – Dynamic image transformation version (used at training).

`__len__`() → int

Method to request data length. Needs re-implementation for linking prepared and desired APIs.

Returns
The number of entries in the dataset.

`existence_tweak`() → None

Tweak to ensure consistency with the class-path structures. Some benchmark have some annotations linking to non-existent images, and this method filters for such entries.

`random_mode`() → None

Turns dynamic mode on. Now random_transform is applied when transform requested.

`static_mode`() → None

Turns static mode on. Now static_transform is applied when transform requested.

`class capsnet.dataset.GermanDataset`(*path_to_img_dir: str*, *path_to_annotations: str*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*)

Bases: `DynamicDataset`

Specifies DynamicDataset for German benchmark.

Variables
`annotation_df` – The Pandas DataFrame with annotations.

__init__(*path_to_img_dir: str*, *path_to_annotations: str*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*) → None

Initializer method.

Parameters

- **path_to_img_dir** – Path to root directory with dataset images.
- **path_to_annotations_file** – Path to file with annotations.
- **static_transform** – Static image transformation version (used at evaluation).
- **random_transform** – Dynamic image transformation version (used at training).

class capsnet.dataset.**RussianDataset**(*path_to_img_dir: str*, *path_to_annotations: str*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*)

Bases: `DynamicDataset`

Specifies DynamicDataset for Russian benchmark.

Variables

annotation_df – The Pandas DataFrame with annotations.

__init__(*path_to_img_dir: str*, *path_to_annotations: str*, *static_transform: Union[None, Compose, Module]*, *random_transform: Union[None, Compose, Module]*) → None

Initializer method.

Parameters

- **path_to_img_dir** – Path to root directory with dataset images.
- **path_to_annotations_file** – Path to file with annotations.
- **static_transform** – Static image transformation version (used at evaluation).
- **random_transform** – Dynamic image transformation version (used at training).

1.1.6 capsnet.estimate module

Implements API related to estimating some static properties from dataset before training.

capsnet.estimate.**estimate_normalization**(*path_to_img_root: str*, *path_to_annotations: Optional[str]*, *DataSetType: Type[DynamicDataset]*, *use_grayscale: bool*, *image_size: Tuple[int, int]*, *estimation_length: int*) → Union[Tuple[float, float], Tuple[List[float], List[float]]]

Estimates mean value and standard deviation by reading a batch of images.

Parameters

- **path_to_img_root** – Path to the folder with images.
- **path_to_annotations** – Path to the file with annotations (if relevant).
- **DataSetType** – The relevant heir of DynamicDataset to handle dataset loading.
- **use_grayscale** – Whether to use grayscale transformation.
- **image_size** – Image shape (2-dimensional).
- **estimation_length** – The amount of images in a batch used for estimating mean and standard deviation.

Returns

Tuple with two entries. The first contains batch mean, the second contains batch standard deviation.

`capsnet.estimate.`**`make_estimating_transformation_base`**(*use_grayscale: bool*, *image_size: Tuple[int, int]*) → Union[None, Compose, Module]

Creates basic image transformation, that may be further composed with other transformations. Essentially, converts to tensor, resizes, and converts to grayscale if requested.

Parameters

- **use_grayscale** – Whether to convert to grayscale.
- **image_size** – Desired image size.

Returns

Compiled and ready-to-use transformation.

1.1.7 capsnet.explain module

Implements API related to explaining the created model.

`class` `capsnet.explain.`**`CapsNetCallable`**(*model: Module*, *normalize_transform: Module*, *use_cuda: bool*)

Bases: `object`

Callable wrapper around CapsNet. The functionality is to apply CapsNet to a batch of raw images (transformed for getting into CapsNet-compatible format) and extract class probabilities.

Variables

- **`use_cuda`** – Whether to use CUDA.
- **`capsnet`** – The trained CapsNet module.
- **`normalize_transform`** – The transformation to apply for converting images into CapsNet-compatible format.

`__call__`(*images: array*) → array

Implementation for the callable method.

Parameters

images – The batch with images to classify with CapsNet.

Returns

Batch with class probabilities.

`__init__`(*model: Module*, *normalize_transform: Module*, *use_cuda: bool*) → None

Initializer method.

Parameters

- **model** – The trained CapsNet module.
- **normalize_transform** – The transformation to apply for converting images into CapsNet-compatible format.
- **use_cuda** – Whether to use CUDA when calling CapsNet.

`capsnet.explain.`**`check_and_make_folder`**(*path_to_dir: str*) → None

Checks if there is a specific folder, and if not, creates the folder.

Parameters
path_to_dir – The path to the directory.

`capsnet.explain.`**`explain_lime`**`(`*benchmark: IBenchmark*, *model: Module*, *use_cuda: bool*, *explanation_dir: str*`)` → None

Explains model with LIME. The results are saved to explanation_dir.

Parameters

- **benchmark** – The benchmark used for training and evaluating the model.
- **model** – The CapsuleNet model to explain.
- **use_cuda** – Whether to use CUDA.
- **explanation_dir** – The directory for saving explanations.

1.1.8 capsnet.keys module

Specifies Keys/Names/Constants.

`class capsnet.keys.`**`BenchmarkName`**

Bases: `object`

Keys/Names for benchmarks.

Variables

- **`CHINESE`** (*`str`*) – Key/Name for Chinese benchmark.
- **`RUSSIAN`** (*`str`*) – Key/Name for Russian benchmark.
- **`BELGIUM`** (*`str`*) – Key/Name for Belgian benchmark.
- **`GERMANY`** (*`str`*) – Key/Name for German benchmark.
- **`ALL`** (*`str`*) – Key/Name for sequentially training on all benchmarks.
- **`CHOICE_OPTIONS`** (*`str`*) – The list with choices for CLI argument parsing.

`ALL: str = 'all'`

`BELGIUM: str = 'belgium'`

`CHINESE: str = 'chinese'`

`CHOICE_OPTIONS: str = ['chinese', 'russian', 'belgium', 'germany', 'all']`

`GERMANY: str = 'germany'`

`RUSSIAN: str = 'russian'`

`class capsnet.keys.`**`ColorSchema`**

Bases: `object`

Keys/Names for color schemas.

Variables

- **`GRAYSCALE`** (*`str`*) – Key for grayscale color schema.
- **`RGB`** (*`str`*) – Key for RGB color schema.

GRAYSCALE: str = 'grayscale'

RGB: str = 'rgb'

class capsnet.keys.**Constants**

Bases: `object`

Some helpful pre-computed constants.

Variables

- **MEAN_CHINESE_GRAYSCALE** (*float*) – Mean value used for normalizing chinese benchmark in grayscale format.
- **STD_CHINESE_GRAYSCALE** (*float*) – Std value used for normalizing chinese benchmark in grayscale format.

MEAN_CHINESE_GRAYSCALE: float = 0.4255

STD_CHINESE_GRAYSCALE: float = 0.2235

class capsnet.keys.**FileFolderPaths**

Bases: `object`

Names and paths for different file/folder locations.

Variables

- **CHINESE_TRAIN_ROOT** (*str*) – Path to root folder with train images for Chinese benchmark.
- **CHINESE_TRAIN_ANNOTATIONS_ROOT** (*str*) – Path to root folder with train annotations for Chinese benchmark.
- **CHINESE_TRAIN_ANNOTATIONS** (*str*) – Path to file with train annotations for Chinese benchmark.
- **CHINESE_TEST_ROOT** (*str*) – Path to root folder with test images for Chinese benchmark.
- **CHINESE_TEST_ANNOTATIONS_ROOT** (*str*) – Path to root folder with test annotations for Chinese benchmark.
- **CHINESE_TEST_ANNOTATIONS** (*str*) – Path to file with test annotations for Chinese benchmark.
- **GERMAN_TRAIN_ROOT** (*str*) – Path to root folder with train images for German benchmark.
- **GERMAN_TRAIN_ANNOTATIONS** (*str*) – Path to file with train annotations for German benchmark.
- **GERMAN_TEST_ROOT** (*str*) – Path to root folder with test images for German benchmark.
- **GERMAN_TEST_ANNOTATIONS** (*str*) – Path to file with test annotations for German benchmark.
- **BELGIUM_TRAIN_ROOT** (*str*) – Path to root folder with train images for Belgian benchmark.
- **BELGIUM_TRAIN_ANNOTATIONS** (*Optional[str]*) – Belgian annotations are included in file names, therefore set to None.
- **BELGIUM_TEST_ROOT** (*str*) – Path to root folder with test images for Belgian benchmark.
- **BELGIUM_TEST_ANNOTATIONS** (*Optional[str]*) – Belgian annotations are included in file names, therefore set to None.
- **RUSSIAN_TRAIN_ROOT** (*str*) – Path to root folder with train images for Russian benchmark.

- RUSSIAN_TRAIN_ANNOTATIONS (*str*) – Path to file with train annotations for Russian benchmark.
- RUSSIAN_TEST_ROOT (*str*) – Path to root folder with test images for Russian benchmark.
- RUSSIAN_TEST_ANNOTATIONS (*str*) – Path fo file with test annotations for Russian benchmark.

BELGIUM_TEST_ANNOTATIONS: Optional[str] = None

BELGIUM_TEST_ROOT: str = '../benchmarks/belgium-TSC/BelgiumTSC_Training/Training'

BELGIUM_TRAIN_ANNOTATIONS: Optional[str] = None

BELGIUM_TRAIN_ROOT: str = '../benchmarks/belgium-TSC/BelgiumTSC_Training/Training'

CHINESE_TEST_ANNOTATIONS: str = '../benchmarks/China-TSRD/TSRD-Test-Annotation\\TsignRecgTest1994Annotation.txt'

CHINESE_TEST_ANNOTATIONS_ROOT: str = '../benchmarks/China-TSRD/TSRD-Test-Annotation'

CHINESE_TEST_ROOT: str = '../benchmarks/China-TSRD/TSRD-Test-Images/'

CHINESE_TRAIN_ANNOTATIONS: str = '../benchmarks/China-TSRD/TSRD-Train-Annotation\\TsignRecgTrain4170Annotation.txt'

CHINESE_TRAIN_ANNOTATIONS_ROOT: str = '../benchmarks/China-TSRD/TSRD-Train-Annotation'

CHINESE_TRAIN_ROOT: str = '../benchmarks/China-TSRD/TSRD-Train-Images/'

GERMAN_TEST_ANNOTATIONS: str = '../benchmarks/german-GTRSD/Test.csv'

GERMAN_TEST_ROOT: str = '../benchmarks/german-GTRSD/'

GERMAN_TRAIN_ANNOTATIONS: str = '../benchmarks/german-GTRSD/Train.csv'

GERMAN_TRAIN_ROOT: str = '../benchmarks/german-GTRSD/'

RUSSIAN_TEST_ANNOTATIONS: str = '../benchmarks/rtsd-r1/gt_test.csv'

RUSSIAN_TEST_ROOT: str = '../benchmarks/rtsd-r1/test/'

RUSSIAN_TRAIN_ANNOTATIONS: str = '../benchmarks/rtsd-r1/gt_train.csv'

RUSSIAN_TRAIN_ROOT: str = '../benchmarks/rtsd-r1/train/'

class capsnet.keys.**NameKeys**

Bases: object

Naming-related keys and templates.

Variables

- BEST_CHECKPOINT (*str*) – Name for the best checkpoint.
- TRAINDIR (*str*) – Name for training folder.
- EXPLANATIONS (*str*) – Name for the explanation folder.
- VISUALIZATIONS (*str*) – Name for the visualizations folder.
- TRAIN_MODE_STRING (*str*) – Name/Key for the training mode.

- TEST_MODE_STRING (*str*) – Name/Key for testing mode.
- SOURCE_PNG (*str*) – Name pattern for source PNG file.
- RESTORED_PNG (*str*) – Name pattern for restored PNG file.
- EXPLAIN_SRC_PNG (*str*) – Name pattern for PNG file storing source image for LIME explaining.
- EXPLAIN_SUPERPIXELS_PNG (*str*) – Name pattern for PNG file storing LIME explanation (superpixels only).
- EXPLAIN_ALL_PNG (*str*) – a Name pattern for PNG file storing LIME explanation (complete image).
- STATS_JSON (*str*) – Name for the file with comparing statistics in JSON format.
- STATS_TEX (*str*) – Name for the file with comparing statistics in LaTeX format.
- STATS_XLSX (*str*) – Name for the file with comparing statistics in Excel format.

BEST_CHECKPOINT: str = 'best'

EXPLAIN_ALL_PNG: str = 'image-{}-explain-all.png'

EXPLAIN_SRC_PNG: str = 'image-{}-explain-src.png'

EXPLAIN_SUPERPIXELS_PNG: str = 'image-{}-explain-superpixels.png'

EXPLANATIONS: str = 'explanations/{}'

RESTORED_PNG: str = 'restored.png'

SOURCE_PNG: str = 'source.png'

STATS_JSON: str = 'stats.json'

STATS_TEX: str = 'stats.tex'

STATS_XLSX: str = 'stats.xlsx'

TEST_MODE_STRING: str = 'test'

TRAINDIR: str = 'traindir'

TRAIN_MODE_STRING: str = 'train'

VISUALIZATIONS: str = 'visualizations/{}'

class capsnet.keys.**StatsTableKeys**

Bases: object

Column names in summary tables.

Variables

- DATASET (*str*) – Names the column with benchmark name.
- EPOCH_ID (*str*) – Names the column with id of optimal epoch.
- TEST_ACCURACY (*str*) – Names the column with test accuracies.
- TRAIN_ACCURACY (*str*) – Names the column with train accuracies.
- TEST_LOSS (*str*) – Names the column with test losses.

- `TRAIN_LOSS` (*str*) – Names the column with train losses.

DATASET: str = 'Benchmark'

EPOCH_ID: str = 'Epoch Id'

TEST_ACCURACY: str = 'Test Accuracy'

TEST_LOSS: str = 'Test Loss'

TRAIN_ACCURACY: str = 'Train Accuracy'

TRAIN_LOSS: str = 'Train Loss'

class capsnet.keys.**TableColumns**

Bases: `object`

Column names in tables.

Variables

- `GERMAN_CLASS_COLUMN` (*str*) – The name of the column containing classes in German benchmark.
- `GERMAN_PATH_COLUMN` (*str*) – The name of the column containing paths in German benchmark.
- `RUSSIAN_CLASS_COLUMN` (*str*) – The name of the column containing classes in Russian benchmark.
- `RUSSIAN_PATH_COLUMN` (*str*) – The name of the column containing paths in Russian benchmark.

GERMAN_CLASS_COLUMN: str = 'ClassId'

GERMAN_PATH_COLUMN: str = 'Path'

RUSSIAN_CLASS_COLUMN: str = 'class_number'

RUSSIAN_PATH_COLUMN: str = 'filename'

1.1.9 capsnet.launch module

API related to launching training experiments and statistics aggregation.

capsnet.launch.**fill_stats**(*json_stats: Dict[str, Any]*, *table_stats: Dict[str, Any]*, *epoch_id: int*, *test_accuracy: float*, *test_loss: float*, *train_accuracy: float*, *train_loss: float*, *benchmark_name: str*) → None

Adds benchmark statistics to stats-collecting structres.

Parameters

- **json_stats** – Storage for statistics in JSON format.
- **table_stats** – Storage for statistics in pre-table format (for creating pandas DataFrame with from_dict API).
- **epoch_id** – The index of optimal epoch.
- **test_accuracy** – The test accuracy.
- **test_loss** – The test loss.

- **train_accuracy** – The train accuracy.
- **train_loss** – The train loss.
- **benchmark_name** – The benchmark name.

capsnet.launch.**output_stats**(*json_stats: Dict[str, Any]*, *table_stats: Dict[str, Any]*) → None

Exports stats to files.

Parameters

- **json_stats** – Statistics in JSON format.
- **table_stats** – Statistics in table-friendly format.

capsnet.launch.**perform_belgium_launches**(*json_stats: Dict[str, Any]*, *table_stats: Dict[str, Any]*) → None

Executes training launch on Belgium benchmark.

Parameters

- **json_stats** – Storage for statistics in JSON format.
- **table_stats** – Storage for statistics in pre-table format (for creating pandas DataFrame with from_dict API).

capsnet.launch.**perform_chinese_launches**(*json_stats: Dict[str, Any]*, *table_stats: Dict[str, Any]*) → None

Executes training launch on Chinese benchmark.

Parameters

- **json_stats** – Storage for statistics in JSON format.
- **table_stats** – Storage for statistics in pre-table format (for creating pandas DataFrame with from_dict API).

capsnet.launch.**perform_german_launches**(*json_stats: Dict[str, Any]*, *table_stats: Dict[str, Any]*) → None

Executes training launch on German benchmark.

Parameters

- **json_stats** – Storage for statistics in JSON format.
- **table_stats** – Storage for statistics in pre-table format (for creating pandas DataFrame with from_dict API).

capsnet.launch.**perform_launches**(*args: Namespace*) → None

Performs experimetns and collects summary statistics.

Parameters

args – Namespace with CLI arguments.

capsnet.launch.**perform_russian_launches**(*json_stats: Dict[str, Any]*, *table_stats: Dict[str, Any]*) → None

Executes training launch on Russian benchmark.

Parameters

- **json_stats** – Storage for statistics in JSON format.
- **table_stats** – Storage for statistics in pre-table format (for creating pandas DataFrame with from_dict API).

capsnet.launch.**perform_test_launches**(*json_stats: Dict[str, Any]*, *table_stats: Dict[str, Any]*) → None

Executes test launch.

Parameters

- **json_stats** – Storage for statistics in JSON format.
- **table_stats** – Storage for statistics in pre-table format (for creating pandas DataFrame with from_dict API).

1.1.10 capsnet.layers module

Implements modules and layers used for building Capsule Network.

`class capsnet.layers.`**`AgreementRouting`**`(`*agreement_config:* ConfigAgreement, *squash_config:* ConfigSquash`)`

Bases: `Module`

Implements routing agreement. This process updates coupling coefficients between primary capsule layers through several iterations, and returns capsule outputs after routing with iterative coupling.

Variables

- **`n_iterations`** – Number of routing iterations.
- **`num_input_caps`** – Number of capsules in the input layer.
- **`num_output_caps`** – Number of capsules in the output layer.
- **`use_cuda`** – Whether to use CUDA (true/false).
- **`squash`** – The squashing module to apply while routing.

`__init__``(`*agreement_config:* ConfigAgreement, *squash_config:* ConfigSquash`)` → None

Initializer method.

Parameters

- **agreement_config** – Configuration for routing agreement module.
- **squash_config** – Configuration for squashing module.

`forward``(`*u_ji_predict_5d: Union[FloatTensor, FloatTensor]*`)` → Union[FloatTensor, FloatTensor]

Applies routing agreement.

Parameters

u_ji_predict_5d – The prediction vectors (obtained by multiplying capsule outputs u_i by weights W_{ij}).

Returns

Outputs v_j of the capsules for the next capsule layer (squashed).

`class capsnet.layers.`**`ConvLayer`**`(`*config:* ConfigConv`)`

Bases: `Module`

Applies convolution and batch normalization (if configured), with ReLU activation afterwards.

Variables

- **`conv`** – The convolution module from PyTorch, configured.
- **`batch_norm`** – The batch normalization module from PyTorch, configured.
- **`use_batch_norm`** – Boolean flag specifying whether batch norm is applied.

__init__(*config: ConfigConv*) → None

Initializer method. Configures batch normalization and convolution.

Parameters
config – The configuration object for convolution layer.

forward(*input_tensor: Union[FloatTensor, FloatTensor]*) → Union[FloatTensor, FloatTensor]

Applies convolution layer to input_tensor. Then applies batch normalization, if configured, and ReLU activation on top.

Parameters
input_tensor – Input tensor to convolution.

Returns
The tensor after applying the announced transformations.

class capsnet.layers.**PrimaryCaps**(*primary_config: ConfigPrimary*, *squash_config: ConfigSquash*)

Bases: `Module`

Implements primary capsules. Primary capsules are the first layer of capsules applied. Capsules are interpreted as something for finding important objects and measuring properties of those objects. The length of the capsule output corresponds to the probability to have the object in the input. Deeper capsules account for deeper objects.

Variables

- **capsules** – A module list with configured convolution layers applied at each capsule.
- **hook_handles** – List with hooks/tricks to improve learning, e.g. replacing NaNs with zeroes in gradients for stabilization.
- **dropouts** – A module list with configured dropouts to be applied at capsule convolution outputs if configured.
- **capsule_output_dim** – The flattened output dimension the capsule.
- **squash** – Module for squashing pre-squashed stacked capsule outputs.
- **use_dropout** – Boolean flag specifying whether dropout should be applied.

__init__(*primary_config: ConfigPrimary*, *squash_config: ConfigSquash*) → None

Initializer method. Configures convolutions and dropouts for each capsule, creates squashing module and adds gradient hooks (if requested in config).

Parameters

- **primary_config** – Configuration for primary capsules.
- **squash_config** – Configuration for squashing module.

forward(*input_tensor: Union[FloatTensor, FloatTensor]*) → Union[FloatTensor, FloatTensor]

Applies primary capsules. Convolves, applies dropout (if specified by self.use_dropout), stacks, and squashes.

Parameters
input_tensor – The input tensor to the primary capsule layer.

Returns
Squashed capsule outputs.

remove_hooks() → None

Deactivates all applied hooks.

class `capsnet.layers.`**RecognitionCaps**(*recognition_config:* ConfigRecognition, *agreement_config:* ConfigAgreement, *squash_config:* ConfigSquash)

Bases: `Module`

Implements recognition capsules. Recognition capsules are those applied after the first layer with primary capsules. Capsules are interpreted as something for finding important objects and measuring properties of those objects. The length of the capsule output corresponds to the probability to have the object in the input. Deeper capsules account for deeper objects.

Variables

- **`num_input_caps`** – The number of input capsules.
- **`input_caps_dim`** – Dimension of the input capsule vector.
- **`num_output_caps`** – The number of output capsules.
- **`output_caps_dim`** – Dimension of output capsule vector.
- **`W_matrix_5d`** – Weight matrix to compute prediction vectors u[j|i] from input capsule outputs u_i.
- **`W_hook_handle`** – Handle for W-related gradient hook. None if no hook applied.
- **`agreement_routing`** – Configured module to apply agreement routing.
- **`dropout`** – Configured module to apply dropout.
- **`use_dropout`** – Boolean flag whether to apply dropout regularization.

`__init__`(*recognition_config:* ConfigRecognition, *agreement_config:* ConfigAgreement, *squash_config:* ConfigSquash) → None

Initializer method. Creates weight matrix W_matrix_5d and corresponding gradient hook (if configured). Also configures agreement_routing and dropout.

Parameters

- **recognition_config** – Configuration for recognition capsule.
- **agreement_config** – Configuration for agreement routing module.
- **squash_config** – Configuration for squashing module.

`forward`(*input_tensor: Union[FloatTensor, FloatTensor]*) → Union[FloatTensor, FloatTensor]

Applies recognition capsule to input_tensor. Essentially, converts outputs of previous capsule layer to ouptut of the next capsule layer.

Parameters

input_tensor – The tensor with outputs from the previous capsule layer.

Returns

The tensor with outputs for the further capsule layer.

`remove_hooks`() → None

Deactivates all applied hooks.

class `capsnet.layers.`**ReconstructionNet**(*config:* ConfigReconstruction)

Bases: `Module`

Module to reconstruct the image of the predicted class (i.e. class with longest capsule vector). This reconstruction is used in the part of loss function computation (as regularizer), and can also provide an opportunity to create reconstructions for visual examination.

Variables

- `reconstruction_layers` – A sequence with reconstruction layers. Each layer in the sequence is fully-connected with ReLU on top (except the final layer, where the result is activated with Sigmoid).
- `use_cuda` – Whether to use CUDA.
- `num_output_channels` – Number of channels in the output image.
- `output_image_size` – The size of ouptut image.
- `num_classes` – The number of classes.

`__init__`(*config:* ConfigReconstruction) → None

Initializer method. Creates reconstruction layers.

Parameters
config – Configuration for the reconstruction module.

`forward`(*input_tensor: Union[FloatTensor, FloatTensor]*) → Tuple[Union[FloatTensor, FloatTensor], Union[FloatTensor, FloatTensor]]

Applies reconstruction module.

Parameters
input_tensor – The result from final capsule layer.

Returns
Tuple with two elements. The first contains reconstructions, the second contains the mask for the selected class.

`class capsnet.layers.SquashLayer`(*config:* ConfigSquash)

Bases: `Module`

Implements squashing. The formula from original paper is:
$v_j = ||s_j||^2 / (1 + ||s_j||^2) * s_j / ||s_j||$ However, due to numerical stability issues, the applied formula is: $v_j = (||s_j + eps_input||^2 + eps_norm) / (eps_denom + 1 + eps_norm + ||s_j + eps_input||^2) * (s_j + eps_input) / sqrt(eps_norm + ||s_j + eps_input||^2 + eps_sqrt)$

Variables

- `eps_denom` – Shift added in denominator for numerical stability.
- `eps_sqrt` – Shift added under square root for numerical stability.
- `eps_input` – Shift for input tensor added for numerical stability.
- `eps_norm` – Shift for squared norm added for numerical stability.

`__init__`(*config:* ConfigSquash) → None

Initializer method.

Parameters
config – Configuration for the squash layers. Specifies attributes.

`forward`(*input_tensor: Union[FloatTensor, FloatTensor]*) → Union[FloatTensor, FloatTensor]

Applies squashing.

Parameters
input_tensor – The tensor to squash.

Returns
The squashed tensor.

capsnet.layers.**fix_nan_gradient_hook**(*gradient: Union[FloatTensor, FloatTensor]*) → Optional[Union[FloatTensor, FloatTensor]]

Replaces nans with zeroes in PyTorch gradient.

Parameters

gradient – The gradient tensor to process.

Returns

None if no NaN values is found in the gradient. Otherwise, returns gradient with NaNs replaced by zeroes.

capsnet.layers.**nan_gradient_hook_module**(*module: Module*, *in_gradient: Union[FloatTensor, FloatTensor]*, *out_gradient: Union[FloatTensor, FloatTensor]*) → Optional[List[Union[FloatTensor, FloatTensor]]]

Trick to remove nan values from gradients. The format of the argument is to support PyTorch API; in reality, only out_gradient is updated. Iterates through all members in out_gradient and applies fix_nan_gradient_hook method.

Parameters

- **module** – The relevant module to process.
- **in_gradient** – The input gradient of the module to process.
- **out_gradient** – The output gradient of the module to process.

Returns

None if nothing is updated/rewrited. Outherwise, returns list with updated gradients.

1.1.11 capsnet.network module

Implements Capsule Network for images classification.

class capsnet.network.**CapsNet**(*config:* ConfigNetwork)

Bases: `Module`

Implements Capsule Network for image classification. This architectures starts by applying convolution layers, then uses primary capsules and applies recognition capsules afterwards. The loss function consists of two components: margin_loss and reconstruction_loss. Margin loss estimates the quality of the actual classification, while reconstruction loss regularizes on the difference between reconstructed class and original image.

Variables

- **conv_layer** – The initial convolution layer, pre-processing for input images.
- **primary_capsules** – The primary capsule layer.
- **recognition_capsules** – The layer with recognition capsules.
- **reconstruction_net** – Module for reconstructing the most probable class.
- **net_reconstruction_loss_reg** – Regularization coefficient for reconstruction loss.
- **net_margin_loss_blend** – Coefficient to blend impact of lower and upper margin parts.
- **net_margin_upper** – Upper margin in the margin loss.
- **net_margin_lower** – Lower margin in the margin loss.
- **use_square_in_margin_loss** – Whether to use squared margins in margin loss computation.
- **mse_loss** – Module to compute MSE loss.

`__init__`(*config:* ConfigNetwork) → None

Initializer method. Prepares conv_layer, primary_capsules, recognition_capsules, reconstruction_net and margin_loss.

Parameters
config – Configuration for Capsule Network.

`forward`(*input_tensor: Union[FloatTensor, FloatTensor]*) → Tuple[Union[FloatTensor, FloatTensor], Union[FloatTensor, FloatTensor], Union[FloatTensor, FloatTensor], Union[FloatTensor, FloatTensor]]

Applies capsule network to input tensor (batch with images).

Parameters
input_tensor – The batch with images.

Returns
Tuple with four tensors. The first is the output from final capsules, the second is the reconstructed most probable class, the third is the mask with predicted class, the fourth contains class probabilities.

`loss`(*data_tensor: Union[FloatTensor, FloatTensor]*, *output_tensor: Union[FloatTensor, FloatTensor]*, *target_tensor: Union[FloatTensor, FloatTensor]*, *reconstructions_tensor: Union[FloatTensor, FloatTensor]*) → Union[FloatTensor, FloatTensor]

Method to compute loss function.

Parameters

- **data_tensor** – The tensor with input batch.
- **output_tensor** – The tensor with capsule outputs.
- **target_tensor** – Target classes.
- **reconstructions_tensor** – The tensor with reconstructed batch.

Returns
Tensor with loss function.

`margin_loss`(*capsule_tensor: Union[FloatTensor, FloatTensor]*, *target_mask: Union[FloatTensor, FloatTensor]*) → Union[FloatTensor, FloatTensor]

Method to compute margin loss.

Parameters

- **capsule_tensor** – Tensor with capsule outputs.
- **target_mask** – Mask with true target labels.

Returns
Tensor with average margin loss.

`reconstruction_loss`(*data_tensor: Union[FloatTensor, FloatTensor]*, *reconstructions_tensor: Union[FloatTensor, FloatTensor]*) → Union[FloatTensor, FloatTensor]

Method to compute reconstruction loss.

Parameters

- **data_tensor** – Tensor containing batch with original images.
- **reconstructions_tensor** – Tensor containing batch with reconstructed images.

Returns
Tensor with average reconstruction loss, multiplied by regularization coefficient.

`remove_hooks()` → None

Removes hooks. Hooks are tricks to simplify/fix learning, but can slow down the inference process.

1.1.12 capsnet.run_capsnet module

Entry point. Parses CLI arguments, setups logging and calls API to perform experiments.

`capsnet.run_capsnet.parse_arguments()` → Namespace

Defines and parses CLI arguments.

Returns

Namespace with parsed CLI arguments.

`capsnet.run_capsnet.setup_logging()` → None

Setups logging format.

1.1.13 capsnet.train module

Implements CapsNet training logic.

`capsnet.train.cuda_cache_reset`(*use_cuda: bool*) → None

Explicitly called memory management.

Parameters

use_cuda – Whether CUDA is used, since then CUDA-specific memory management is requested.

`capsnet.train.do_training`(*setup_config:* SetupConfig) → Tuple[int, float, float, float, float]

Entry point to the training procedure. Trains capsule network, creates visualizations and provides LIME explanations if requested.

Parameters

setup_config – Experiment configuration.

Returns

Tuple with five elements. The first is the epoch number that achieved the best test performance. The second is the test accuracy on that epoch. The third is the test loss value on that epoch. The fourth is the train accuracy on that epoch. The fifth is the train loss value on that epoch.

`capsnet.train.dump_checkpoint`(*training_config:* ConfigTraining, *benchmark_name: str*, *checkpoint_id: str*, *epoch_idx: int*, *capsule_net: Module*, *optimizer: Optimizer*, *test_loss: float*, *test_accuracy: float*, *train_loss: float*, *train_accuracy: float*) → None

Dumps checkpoint.

Parameters

- **training_config** – The training configuration.
- **benchmark_name** – The name of the benchmark.
- **checkpoint_id** – The checkpoint identifier.
- **epoch_idx** – The epoch index.
- **capsule_net** – The network at the current state.
- **optimizer** – The optimizer at the current state.
- **test_loss** – The loss value on test set.

- **test_accuracy** – The accuracy value on test set.
- **train_loss** – The loss value on train set.
- **train_accuracy** – The accuracy value on train set.

`capsnet.train.`**`eval_epoch`**`(`*epoch_idx: int*, *benchmark:* IBenchmark, *capsule_net: Module*, *use_cuda: bool*, *n_classes: int*, *writer: SummaryWriter*, *log_frequency: int*`)` → Union[Tuple[float, float], Tuple[Union[FloatTensor, FloatTensor], Union[FloatTensor, FloatTensor], float, float]]

Evaluates epoch.

Parameters

- **epoch_idx** – Epoch index.
- **benchmark** – Benchmark of the current experiment.
- **capsule_net** – The capsule network for evaluation.
- **use_cuda** – Whether to use CUDA.
- **n_classes** – Number of classes.
- **writer** – Writer to TensorBoard.
- **log_frequency** – Frequency of logging.

Returns

Tuple with four elements. The first is some testing batch with data. The second is the reconstructions on that batch. The third is the average test loss value on that epoch. The fourth is the test accuracy on that epoch.

`capsnet.train.`**`iterate_training`**`(`*start_epoch_idx: int*, *n_epochs: int*, *benchmark:* IBenchmark, *capsule_net: Module*, *optimizer: Optimizer*, *use_cuda: bool*, *n_classes: int*, *writer: SummaryWriter*, *training_config:* ConfigTraining, *benchmark_name: str*`)` → Tuple[Union[FloatTensor, FloatTensor], Union[FloatTensor, FloatTensor], int, float, float, float, float]

Method responsible for iterative training. Iterates until the desired number of epochs is achieved.

Parameters

- **start_epoch_idx** – The epoch intex to start.
- **n_epochs** – The total number of epochs.
- **benchmark** – The current benchmark for training and evaluation.
- **capsule_net** – The network to train and evaluate.
- **optimizer** – The optimizer for loss optimization.
- **use_cuda** – Whether to use CUDA.
- **n_classes** – The number of classes.
- **writer** – Writer for providing information to TensorBoard.
- **training_config** – The training configuration.
- **benchmark_name** – The name of the current benchmark for training and evaluation.

Returns

Tuple with seven elements. The first is some testing batch with data. The second is the reconstructions on that batch. The third is the epoch number that achieved the best test performance.

The fourth is the test accuracy on that epoch. The fifth is the test loss value on that epoch. The sixth is the train accuracy on that epoch. The seventh is the train loss value on that epoch.

`capsnet.train.`**`process_epoch`**`(`*epoch_idx: int*, *benchmark:* IBenchmark, *capsule_net: Module*, *optimizer: Optional[Optimizer]*, *use_cuda: bool*, *n_classes: int*, *writer: SummaryWriter*, *train_mode: bool*, *log_frequency: int*, *use_clipping: bool*, *clip_value: Optional[float]*`)` → Union[Tuple[float, float], Tuple[Union[FloatTensor, FloatTensor], Union[FloatTensor, FloatTensor], float, float]]

Broad method responsible for epoch training or epoch testing, depending on the input parameters.

Parameters

- **epoch_idx** – Epoch index.
- **benchmark** – Benchmark of the current experiment.
- **capsule_net** – The capsule network for training or evaluation.
- **optimizer** – The optimizer for updating network weights, if relevant.
- **use_cuda** – Whether to use CUDA.
- **n_classes** – Number of classes.
- **writer** – Writer to TensorBoard.
- **train_mode** – If true, train logic is applied. If false, testing logic is applied.
- **log_frequency** – Frequency of logging.
- **use_clipping** – Whether to use gradient clipping.
- **clip_value** – The threshold for gradient clipping.

Returns

A tuple with two or four elements, depending on train_mode – If training, returns a tuple with two elements. The first is the average epoch training loss. The second is training epoch acuracy. If testing, returns a tuple with four elements. The first is some testing batch with data. The second is the reconstructions on that batch. The third is the average test loss value on that epoch. The fourth is the test accuracy on that epoch.

`capsnet.train.`**`train_epoch`**`(`*epoch_idx: int*, *benchmark:* IBenchmark, *capsule_net: Module*, *optimizer: Optimizer*, *use_cuda: bool*, *n_classes: int*, *writer: SummaryWriter*, *log_frequency: int*, *use_clipping: bool*, *clip_threshold: Optional[float]*`)` → Union[Tuple[float, float], Tuple[Union[FloatTensor, FloatTensor], Union[FloatTensor, FloatTensor], float, float]]

Performs epoch of training.

Parameters

- **epoch_idx** – Epoch index.
- **benchmark** – Benchmark of the current experiment.
- **capsule_net** – The capsule network for training.
- **optimizer** – The optimizer for updating network weights.
- **use_cuda** – Whether to use CUDA.
- **n_classes** – Number of classes.
- **writer** – Writer to TensorBoard.
- **log_frequency** – Frequency of logging.

- **use_clipping** – Whether to use gradient clipping.
- **clip_threshold** – The threshold for gradient clipping.

Returns

Tuple with two elements. The first is the average epoch training loss. The second is training epoch acuracy.

1.1.14 capsnet.visualize module

Visualization API.

`capsnet.visualize.plot_images_separately`(*images: array*, *figure_name: str*, *n_images: int*) → None

Plots a sequence of images on the same figure, and saves result to a file.

Parameters

- **images** – The array with images.
- **figure_name** – The path to save figure.
- **n_images** – Number of images.

1.1.15 Module contents

This module implements Car Sign recognition with Capsule Network.

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX