Adaptive Vision

Intuitive

Powerful

Adaptable

Future Processing Sp. z o. o.
ul. Bojkowska 37a, 44-100 Gliwice, Polska
Regon: 278263445 VatID: PL6342532128
Tel. +48 32 461 23 30
E-mail: info@adaptive-vision.com

Adaptive Vision Library 4.10

Introduction

Created: 24.08.2018

Product version: 4.10.2.62669

Table of content:

Deep Learning Training API

Table of contents:

1. Overview

Whole API declaration is located in `DeepLearning/Api.h` file (in `AVL_PATH410\include`) under `avl::DeepLearning` namespace. This namespace contains the following namespaces:

- `AnomalyDetection_Local`, grouping together classes and functions related to training in Anomaly Detection – Local mode,
- `AnomalyDetection_Global`, analogous to the above,
- `AnomalyDetection2`, analogous to the above,
- `FeatureDetection`, analogous to the above,
- `ObjectClassification`, analogous to the above,
- `InstanceSegmentation`, analogous to the above,
- and `Common`, grouping together classes and functions used in multiple namespaces mentioned before.

Each namespace (except `Common`) contains almost the same set of types and functions – in respect of their functionality and purpose. Naturally, types and functions in one namespace differ from their counterparts from another namespaces in respect of their signatures and declaration in code.

Example usage of Deep Learning Training API in Feature Detection mode is shown in *AVL_EXAMPLES_DIR/08 Deep Learning/02 Training - Feature Detection* project.

2. Types

Namespaces except `Common`, declare classes:

- `PreprocessingConfig`, containing setters and getters for preprocessing settings (e.g. downsample).
- `AugmentationsConfig`, containing setters and getters for augmentations settings (e.g. flips, rotation).
- `TrainingConfig`, containing setters and getters for general training settings (e.g. iteration count). As well as pointers to objects of types mentioned above.
- `Sample`, containing setters and getters for one training sample (e.g. path to image file).
- `TrainingEventsHandler`, containing virtual methods used as handlers for various events happening during training process and solving training samples. This class is intended for inheriting. Handling events is described in detail in section Handling events

In addition, each class (except `TrainingEventsHandler`) has 2 more methods:

- `Create(...)` - static method intended for constructing objects of specific type. It allows setting all fields at the same time, without calling multiple setters after construction.
- `Clone()` - method for creating new objects being the exact copy of cloned one.

Due to differences between supported network types and, consequently, different sets of possible parameters, mentioned classes may differ from theirs counterparts from other namespaces. Classes except `TrainingEventsHandler`are **not** intended for being inherited by user types.

`Common`namespace contains 2 classes important for user:

- `ModelInfo`- containing information about already existing in training directory model file. Currently, only validation history is provided. It is used in `ReceivedExistingModelInfo(...)` event handler.

- `Progress`- containing data about progress in training process or solving training samples. Progress information is divided into 3 fields:
  - `Stage`- very general information describing process advancement.
  - `Phase`- more fine-grained information of advancement in current `Stage`. It contains 2 integers: total number of phases and current phase. Phases does not have take the same time to finish.
  - and optional `Step`- some `Phases`can be divided into smaller steps. In such cases, this field contains 2 integers: total number of steps in current `Phase`and number of already finished steps. Each step should take roughly the same time to finish.

  These fields are useful for creating progress bars and so on.

  Apart from that, `Progress`objects contain information about current training, validation values (if applicable) and boolean value indicating that validation has started.

3. Functions

Namespaces except `Common`, declare functions:

CreateSamples

Helper function intended for creating array of training samples from images in given directory with given parameters. Set of these additional parameters depends on mode. See documentation in source code for further explanation.

Syntax

```
atl::Array<std::unique_ptr<Sample>> CreateSamples
(
 ...,
 const atl::String& mask
)
```

Parameters

	Name	Type	Default	Description
➡	...	...	...	Sample parameters, e.g. path to directory with roi images and so on.
➡	mask	const String&	*	Mask used for filtering found files. By default, no files are filtered out. See documentation of FindFiles filter for more information

Train

Performs training process. Returns "true" if model was saved.

Syntax

```
bool Train
(
 DeepLearningConnectionState& state,
 const atl::Array<std::unique_ptr<Sample>> trainingSamples,
 const TrainingConfig& config,
 TrainingEventsHandler& eventsHandler
)
```

Parameters

	Name	Type	Description
➡	state	DeepLearningConnectionState&	Object maintaining connection with service
➡	trainingSamples	const Array<std::unique_ptr<Sample>>&	Array of training samples
➡	config	const TrainingConfig&	Training configuration
➡	eventsHandler	TrainingEventsHandler&	Training events handler. It can be object of `TrainingEventsHandler`or, more common, object of user type inherited from it

Remarks

- This function has overload without `eventsHandler`parameter. It uses object of `TrainingEventsHandler`type instead.
- In Anomaly Detection modes all training samples are automatically solved after training. In other modes, this can be done by SolveTrainingSamples. After solving each sample, `SolvedTrainingSample`event is called.

SolveTrainingSamples

Solves given training samples.

Syntax

```
void SolveTrainingSamples
(
 DeepLearningConnectionState& state,
 const atl::Array<std::unique_ptr<Sample>>& trainingSamples,
 const TrainingConfig& config,
 ...,
 TrainingEventsHandler& eventsHandler
)
```

Parameters

	Name	Type	Description
➡	state	DeepLearningConnectionState&	Object maintaining connection with service
➡	trainingSamples	const Array<std::unique_ptr<Sample>>&	Array of training samples
➡	config	const TrainingConfig&	Training configuration
➡	...	...	Additional parameters depending on mode
➡	eventsHandler	TrainingEventsHandler&	Training events handler. It can be object of `TrainingEventsHandler`or, more common, object of user type inherited from it

Remarks

- After solving each sample, `SolvedTrainingSample`event is called.
- This function does not exists in Anomaly Detection modes since solving training samples is done automatically in Train
- Unlike Train, there is no overload without `TrainingEventsHandler`object - it would be pointless as this function calls `SolvedTrainingSample`event handler and, by default, this handler does nothing.
- Additional parameters, if present, should be described in documentation for corresponding filter (e.g. DeepLearning_SegmentInstances in case of Instance Segmentation mode).

GetWorstValidationValue

Returns worst possible validation value.

Syntax

```
float GetWorstValidationValue()
```

Remarks

- It is useful for initialization validation value (e.g. in custom training events handler) and eliminates need of using special values in comparisons.
- This function does not exists in in Anomaly Detection 2 mode as it would be pointless. This is due significant differences in training process in Anomaly Detection 2 mode comparing to other modes.

IsValidationBetter

Returns "true" if `newValidationValue`is "better" than `oldValidationValue`. Comparing validation values with relational operators is strongly discouraged due the fact that in some modes lower values are better, but in other modes – otherwise.

Syntax

```
bool IsValidationBetter
(
 float oldValidationValue,
 float newValidationValue
)
```

Parameters

	Name	Type	Description
➡	oldValidationValue	float	Base validation value
➡	newValidationValue	float	Compared validation value

Remarks

- This function does not exists in in Anomaly Detection 2 mode as it would be pointless. This is due significant differences in training process in Anomaly Detection 2 mode comparing to other modes.

FindBestValidation

Returns best validation value from array.

Syntax

```
float FindBestValidation
(
 const atl::Array<float>& validationValues
)
```

Parameters

	Name	Type	Description
➡	validationValues	const Array<float>&	Array of validation values

Remarks

- Useful for obtaining best validation value in already existing model in `ReceivedExistingModelInfo` event.
- This function does not exists in in Anomaly Detection 2 mode as it would be pointless. This is due significant differences in training process in Anomaly Detection 2 mode comparing to other modes.

4. Handling events

To communicate with user during training and solving training samples, several events are used. All event handlers are implemented as virtual methods of `TrainingEventsHandler` allowing users to write custom handlers by overriding them.
There are 5 possible events:

- `ReceivedExistingModelInfo(...)` - this handler is called after receiving information about already existing model right after training start. If no model is present at given location, handler is not called. This method takes existing model information as parameter (of type `Common::ModelInfo`) and by default does nothing.
- `ReceivedProgress(...)` - this handler is called after receiving progress information during training and solving training samples. This method takes progress information as parameter of type `Common::Progress` and have to return `true` if process should be stopped or `false` otherwise. Default handler of this event does nothing and do not interrupt process.
- `SavedModelAutomatically(...)` - after training, model file can be saved or discarded. In some cases, Deep Learning Service can make this decision automatically. This handler is called in such cases. This method contains argument indicating whether file was saved or not. Default handler does nothing.
- `SaveModel()` - in most cases, Deep Learning Service cannot determine whether model file should be saved or not. This handler is called in such cases. This method has no arguments but have to return `true` if file should be replaced or `false` otherwise. This decision can be made on the basis of data collected in `ReceivedExistingModelInfo(...)` and `ReceivedProgress(...)` event handlers. Default handler always returns `true` which results in Deep Learning Service saving model file.
- `SolvedTrainingSample(...)` - this handler is called each time after solving training sample. This happens in `Train(...)` (in both AnomalyDetection modes) or `SolveTrainingSamples(...)` (in other modes) functions. This method takes training sample (of type `Sample`) and solution results as arguments. Unlike previous methods, signature of this methods differs between various namespaces. Default handler does nothing.