Reference Manual Road Sign API and Filter Management Library

Generated by Doxygen 1.8.13

Contents

Chapter 1

Einleitung

Die Filterverwaltungs-Bibliothek ist nach dem Pipes-and-Filters-Prinzip aufgebaut. Dieses ist ein gängiges Architekturmuster für Systeme, die Datenströme verarbeiten. In unserem Fall bestehen diese aus (einer Folge von) Einzelbildern mit einem Header, in dem Zusatzinformationen Platz finden. Ein Filter stellt dabei einen Verarbeitungsschritt dar, wobei jeder Filter der Pipe ein Bild entgegennimmt und auch wieder ein Bild herausgibt. Darüber hinaus kann jeder Filter dem Datenpaket bestimmte Metainformationen beifügen, die später für eine genauere Analyse verwendet werden können. Wenn beispielsweise ein Filter ein Verkehrszeichen auf einem Bild an einer bestimmten Position detektiert und im darauffolgenden Filter das Verkehrszeichen klassifiziert wird, wird neben dem Zuschneiden des Bildes auf das erkannte Verkehrszeichen auch die Position des Verkehrszeichens im Original als Metainformation gespeichert. Dies ist eine Abwandlung der herkömmlichen Pipes-and-Filters-Architektur. Weiterhin ist es möglich, dass sich die Pipeline nach bestimmten Verarbeitungsschritten in mehrere untergeordnete Pipes aufteilt: Werden auf einem Bild mehrere Verkehrszeichen erkannt, wird der Filter für die Klassifikation für alle Kandidaten aufgerufen (ggf. parallel).

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- CopyingInputStream
  - anonymous_namespace{TensorflowAndroidJNIUtils.cpp}::IfstreamInputStream 25
  - tensorflow::android::LimitingFileInputStream 26
- RoadSignAPI::DetectedSignCombination 17
- RoadSignAPI::DetectedSignDescriptor 20
- FilterManagementLibrary::Logger 28
- FilterManagementLibrary::PipeSystem::PipeFilter 37
  - FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter 93
    - RoadSignAPI::MobilenetV2RoadSignClassificator 31
    - RoadSignAPI::SSDLiteRoadSignDetector 67
  - RoadSignAPI::ClassifiedSignsGrouper 15
  - RoadSignAPI::DetectionBasedImageSlicer 22
  - RoadSignAPI::RoadSignDuplicationDeleter 60
- FilterManagementLibrary::PipeSystem::PipeRegisteredFilters 44
  - RoadSignAPI::RSAPIPipeRegisteredFilters 64
- FilterManagementLibrary::PipeSystem::PipeWorkingDataSet 45
  - RoadSignAPI::RSAPIWorkingDataSet 65
- FilterManagementLibrary::PipeSystem::ProcessingPipeline 45
- RoadSignAPI::RoadSignAPI 51
- FilterManagementLibrary::TensorflowAndroidJNIUtils 72
- FilterManagementLibrary::TFIntegration::TensorflowNNInstance 74
  - FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier 83
- FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription 86
- FilterManagementLibrary::TensorflowOpenCVUtils 88
- FilterManagementLibrary::TFIntegration::TensorflowResultContainer 90
- FilterManagementLibrary::Utilities 99

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 anonymous_namespace{TensorflowAndroidJNIUtils.cpp} Namespace Reference

Classes

- class IfstreamInputStream

6.2 FilterManagementLibrary Namespace Reference

Everything the FilterManagementLibrary provides.

Namespaces

- PipeSystem

  Everything that has to do with the pipes and filters part of the library.
- TFIntegration

Classes

- class Logger

  Logger providing Desktop Linux <-> Android platform independent logging.
- class TensorflowAndroidJNIUtils
- class TensorflowOpenCVUtils
- class Utilities

  Providing some basic utilities, like checking if a file exists.

6.2.1 Detailed Description

Everything the FilterManagementLibrary provides.

6.3 FilterManagementLibrary::PipeSystem Namespace Reference

Everything that has to do with the pipes and filters part of the library.

Classes

- class PipeFilter

  Base class were all filters are derived from.
- struct PipeRegisteredFilters

  Template struct.
- struct PipeWorkingDataSet

  Template structfor PipeWorkingDataSets.
- class ProcessingPipeline

  Pipeline of the pipes and filters architecture.
- class TFNNBasedPipeFilter

  Abstract template class to derive Tensorflow based filters from.

6.3.1 Detailed Description

Everything that has to do with the pipes and filters part of the library.

6.4 FilterManagementLibrary::TFIntegration Namespace Reference

Classes

- class TensorflowNNInstance
- class TensorflowNNInstanceClassifier
- struct TensorflowNNModelDescription
- class TensorflowResultContainer

6.4.1 Detailed Description

Provides functionalitys to load Tensorflow models and use them to do calculations / perform operations, interprete their output and so on.

6.5 google Namespace Reference

Namespaces

- protobuf

6.6 google::protobuf Namespace Reference

6.7 RoadSignAPI Namespace Reference

Classes

- class ClassifiedSignsGrouper
- class DetectedSignCombination
- struct DetectedSignDescriptor
- class DetectionBasedImageSlicer
- class MobilenetV2RoadSignClassificator
- class RoadSignAPI
- class RoadSignDuplicationDeleter
- struct RSAPIPipeRegisteredFilters
- struct RSAPIWorkingDataSet
- class SSDLiteRoadSignDetector

6.8 tensorflow Namespace Reference

Namespaces

- android

6.9 tensorflow::android Namespace Reference

Classes

- class LimitingFileInputStream

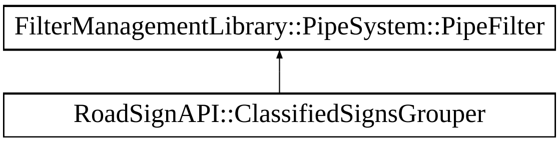
Chapter 7

Class Documentation

7.1 RoadSignAPI::ClassifiedSignsGrouper Class Reference

```
#include <ClassifiedSignsGrouper.h>
```

Inheritance diagram for RoadSignAPI::ClassifiedSignsGrouper:


FilterManagementLibrary::PipeSystem::PipeFilter
RoadSignAPI::ClassifiedSignsGrouper

Private Member Functions

- bool initByPipeSetup ()

  Initializes the filter.
- bool process ()

  The process function of this filter.

Private Attributes

- RSAPIWorkingDataSet ∗ castedWorkingDataSet
- int imageWidth
- float horizontalRangePercentage = 0.025

Additional Inherited Members

7.1.1 Detailed Description

For the user, it is nice to know which signs are mounted onto the same pole, so they are logically grouped. This has a direct effect on the scope of some signs (i.e. signs indicating danger can limit the validity of speed limits for the duration of a dangerous right turn etc.). This Filter aims to group those signs into one DetectedSignCombination}. For this, it estimates where the pole of a sign is located (for more, please refer to the process() function and to DetectedSignCombination}.

7.1.2 Member Function Documentation

7.1.2.1 initByPipeSetup()

```
bool RoadSignAPI::ClassifiedSignsGrouper::initByPipeSetup ( )  [private], [virtual]
```

Initializes the filter.

As this filter does not need any specific environment variables, nothing really exciting is happenning here..

Returns

true, always

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References castedWorkingDataSet, FilterManagementLibrary::PipeSystem::PipeFilter::pipeWorkingDataSet, and FilterManagementLibrary::Logger::printfln().

7.1.2.2 process()

```
bool RoadSignAPI::ClassifiedSignsGrouper::process ( )  [private], [virtual]
```

The process function of this filter.

Iterates through all signs that were successfully classified and whose IDs were thus added to the classifier↩ApprovedSigns list of the RSAPIWorkingDataSet}. It calculates an estimated Position of the pole of a sign (by using the middle of the X coordinates of the edges of the box in which a sign is contained) and adding a threshold to it defined in the local member variable horizontalRangePercentage. which should not be to big! For the first sign in the list, a new DetectedSignCombination} will be created. For all other signs, they are checked against all existing signCombinations (so for the second sign, there is exactly one!). If their X coordinates match the estimated pole area of a DetectedSignCombination), than they are added to it (one sign is only added to one combination!). If no matching combination were found, a new one is created.

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References RoadSignAPI::DetectedSignCombination::addDetectedSign(), castedWorkingDataSet, RoadSign↩API::RSAPIWorkingDataSet::classifierApprovedSigns, RoadSignAPI::RSAPIWorkingDataSet::detectedSign↩Combinations, RoadSignAPI::RSAPIWorkingDataSet::detectedSigns, RoadSignAPI::DetectedSignCombination↩::getGestimatedPolePositionX(), horizontalRangePercentage, imageWidth, FilterManagementLibrary::Pipe↩System::PipeFilter::indicateProcessingFinished(), RoadSignAPI::DetectedSignDescriptor::lowerRight, RoadSign↩API::RSAPIWorkingDataSet::originalImageWidth, and RoadSignAPI::DetectedSignDescriptor::upperLeft.

7.1.3 Member Data Documentation

7.1.3.1 castedWorkingDataSet

```
RSAPIWorkingDataSet* RoadSignAPI::ClassifiedSignsGrouper::castedWorkingDataSet  [private]
```

Just a pointer casted from PipeWorkingDataSet∗ to RSAPIWorkingDataSet∗, so we just don't have to do the casting every time we need it ;)

7.1.3.2 horizontalRangePercentage

```
float RoadSignAPI::ClassifiedSignsGrouper::horizontalRangePercentage = 0.025  [private]
```

We try to find out which signs are on one pole. For this, we take the middle of the box of the detected sign and add a threshold in +X and -X direction, to guess where the pole is. This variable specifies how big this threshold is (i.e. how wide the area is in which the pole is expected). The percentage refers to the total width of the originalBGRImage. If the width for example is 1280, a horizontalPercentageRange of 10% would mean 128 pixels in EACH direction (which, of course would be a lot, so we would use something around 1-5%...). If we would exceed the image boundaries while applying this value, we of course would adjust it to stay within the boundaries (we assert that the range always starts where x > 0 and ends where x < originalBGRImageWidth).

7.1.3.3 imageWidth

```
int RoadSignAPI::ClassifiedSignsGrouper::imageWidth  [private]
```

We store a local copy to the originalImageWidth of the working data set (just for easier access and aesthetically more pleasing code ;))

The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/ClassifiedSignsGrouper.h
- RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/ClassifiedSignsGrouper.cpp

7.2 RoadSignAPI::DetectedSignCombination Class Reference

```
#include <DetectedSignCombination.h>
```

Public Member Functions

- void addDetectedSign (DetectedSignDescriptor detectedSign)

  Adds a sign to the list of signs of this combination.
- int getDetectedSignsAmount () const

  Returns the amount of signs contained by this combination.
- int getGestimatedPolePositionX () const

  Returns the estimated pole position X coordinate of this sign combination.
- const std::vector< DetectedSignDescriptor > ∗ getSignsInCombination () const

  Returns a vector containing all Signs of this combination.
- DetectedSignCombination ()

  Constructor of DetectedSignCombination.
- ∼DetectedSignCombination ()

  Standard Destructor of DetectedSignCombination.

Private Member Functions

- void insertSorted (DetectedSignDescriptor ∗detectedSign)

  Inserts a detectedSign at the right position of this combination.

Private Attributes

- std::vector< DetectedSignDescriptor > signs
- int estimatedPolePositionX

7.2.1 Detailed Description

A class combining multiple DetectedSignDescriptors into one object. The idea is that the RoadSignAPI not only detects and classifies the signs, but also tries to determine which signs form one combination, i.e. which signs are on one pole (this is done by ClassifiedSigsGrouper filter, NOT in this class!). All signs on one pole will be stored in a DetectedSignCombination, which also calculates the estimated X coordinate in the image, where the pole is expected to be by calculating the average of the center X coordinate of all signs in the combination. Furthermore, the signs will be sorted by their y coordinates in from top to bottom (top has the lowest y value, as the coordinate origin is at the top left), what insertion sort is used for (as the signs will be added to the combination one by one).

7.2.2 Constructor & Destructor Documentation

7.2.2.1 DetectedSignCombination()

```
RoadSignAPI::DetectedSignCombination::DetectedSignCombination ( )
```

Constructor of DetectedSignCombination.

Intializes all member variables and constructs a new object of type DetectedSignCombination. Does not do any other specific tasks.

References addDetectedSign(), and signs.

7.2.2.2 ∼DetectedSignCombination()

```
RoadSignAPI::DetectedSignCombination::∼DetectedSignCombination ( )
```

Standard Destructor of DetectedSignCombination.

Standard Destructor of DetectedSignCombination, does not do any specific tasks.

7.2.3 Member Function Documentation

7.2.3.1 addDetectedSign()

```
void RoadSignAPI::DetectedSignCombination::addDetectedSign (
             DetectedSignDescriptor detectedSign )
```

Adds a sign to the list of signs of this combination.

Uses insertSorted(...) to add a detected sign to the list of signs in the this combination, so that the list is sorted by the y values of the signs (from top to bottom)

References estimatedPolePositionX, i, insertSorted(), RoadSignAPI::DetectedSignDescriptor::lowerRight, signs, and RoadSignAPI::DetectedSignDescriptor::upperLeft.

7.2.3.2 getDetectedSignsAmount()

```
int RoadSignAPI::DetectedSignCombination::getDetectedSignsAmount ( ) const
```

Returns the amount of signs contained by this combination.

Returns

int Size of the list (= amount of entries in the list) containing all DetectedSignDescriptor}s of this combination.

References signs.

7.2.3.3 getGestimatedPolePositionX()

```
int RoadSignAPI::DetectedSignCombination::getGestimatedPolePositionX ( ) const
```

Returns the estimated pole position X coordinate of this sign combination.

The estimated pole position X is calculated by forming the average of the center X coordinate (middle of the box in which a sign is contained).

References estimatedPolePositionX, and getSignsInCombination().

7.2.3.4 getSignsInCombination()

```
const std::vector< RoadSignAPI::DetectedSignDescriptor > * RoadSignAPI::DetectedSignCombination↩
::getSignsInCombination ( ) const
```

Returns a vector containing all Signs of this combination.

All DetectedSignCombination}s which were added to this combination will be added to a list, which can be retrieved using this function. This list is sorted by the y values of the signs from top to bottom.

References signs.

7.2.3.5 insertSorted()

```
void RoadSignAPI::DetectedSignCombination::insertSorted (
            DetectedSignDescriptor * detectedSign )  [private]
```

Inserts a detectedSign at the right position of this combination.

Uses insertion sort to add the detected sign to the list of signs in the this combination, so that the list is sorted by the y values of the signs (from top to bottom).

References i, signs, and RoadSignAPI::DetectedSignDescriptor::upperLeft.

7.2.4 Member Data Documentation

7.2.4.1 estimatedPolePositionX

```
int RoadSignAPI::DetectedSignCombination::estimatedPolePositionX  [private]
```

To find out which sign belongs to a group of detected signs (a sign combination), i.e. signs on one pole, an external logic may be interested to know where the "middle" (in x direction) of the signs is. For this, we take all X positions of the signs in a combination, sum them up and calculate the average.

7.2.4.2 signs

```
std::vector<DetectedSignDescriptor> RoadSignAPI::DetectedSignCombination::signs  [private]
```

The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/DetectedSignCombination.h
- RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/DetectedSignCombination.cpp

7.3 RoadSignAPI::DetectedSignDescriptor Struct Reference

```
#include <DetectedSignDescriptor.h>
```

Public Attributes

- cv::Point upperLeft
- cv::Point lowerRight
- int detectionPredictedClassID
- float detectorConfidence
- int classifierApprovedClassID
- float classifierConfidence

7.3.1 Detailed Description

Struct describing position and ID of a roadsign detected and classified in an image. It also contains the confidence values of the detector and the classificator. Mainly used in RSAPIWorkingDataSet.

7.3.2 Member Data Documentation

7.3.2.1 classifierApprovedClassID

```
int RoadSignAPI::DetectedSignDescriptor::classifierApprovedClassID
```

Class ID the classifier determined.

7.3.2.2 classifierConfidence

```
float RoadSignAPI::DetectedSignDescriptor::classifierConfidence
```

Confidence of the classifier for the classification of this detection. Will be filled when the DetectedSignDescriptor is passed to the MobilenetV2RoadSignClassificator.

7.3.2.3 detectionPredictedClassID

```
int RoadSignAPI::DetectedSignDescriptor::detectionPredictedClassID
```

Class ID the detection filter predicts.

7.3.2.4 detectorConfidence

```
float RoadSignAPI::DetectedSignDescriptor::detectorConfidence
```

Confidence of the detector for this detection.

7.3.2.5 lowerRight

```
cv::Point RoadSignAPI::DetectedSignDescriptor::lowerRight
```

Lower right position of the detection of the roadsign (mapped to the coordinates of the original image that was provied, not to the scaled image the detector uses).

7.3.2.6 upperLeft

```
cv::Point RoadSignAPI::DetectedSignDescriptor::upperLeft
```

Upper left position of the detection of the roadsign (mapped to the coordinates of the original image that was provied, not to the scaled image the detector uses).

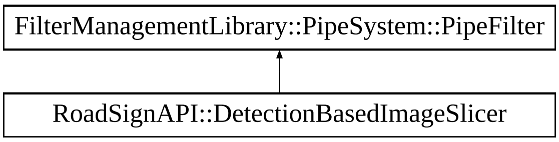
The documentation for this struct was generated from the following file:

- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/DetectedSignDescriptor.h

7.4 RoadSignAPI::DetectionBasedImageSlicer Class Reference

```
#include <DetectionBasedImageSlicer.h>
```

Inheritance diagram for RoadSignAPI::DetectionBasedImageSlicer:


FilterManagementLibrary::PipeSystem::PipeFilter
RoadSignAPI::DetectionBasedImageSlicer

Private Member Functions

- void expandBox (cv::Point ∗upperLeft, cv::Point ∗lowerRight)
- bool initByPipeSetup ()

  Initializes the filter.
- bool process ()

  The process function of this filter.

Private Attributes

- RSAPIWorkingDataSet ∗ castedWorkingDataSet
- float expandPercentage = 0.1
- int minExpandPixels = 5

Additional Inherited Members

7.4.1 Detailed Description

This filter will use the detections the SSDLiteRoadSignDetector provides to cut the originalBGRImage into smaller pieces, which the MobilenetV2RoadSignClassificator can use to classify the detections.

The dataset which was used to train the network models consists of images where each road sign was labeled (using a box which enclosed the sign). These boxes were set in such a way that they perfectly enclose the signs. However to train the classificator, we expaneded each box in every direction (x, -x, y, -y) by 10% of the box width / height or at least 5 pixels (depending on which value is bigger), in order to achieve a greater independence from the background of the road sign and noise. Thus the network learned the road sign's class is independent from it's background. To achieve optimal results, we have to do the same expansion here instead of only crop the images from the given detection before feeding them into the classificator. We use the same values here, stored in expandPercentage (10%) and minExpandPixels (5).

7.4.2 Member Function Documentation

7.4.2.1 expandBox()

```
void RoadSignAPI::DetectionBasedImageSlicer::expandBox (
            cv::Point * upperLeft,
            cv::Point * lowerRight ) [private]
```

Expands the given coordinates of the given box (rectangle) by expandPercentage (see member variables) in each direction, if the amount of pixels to expand is bigger than minExpandPixels (see member variables), otherwise minExpandPixels will be used as amount of pixels to expand. If we exceed the border of the originalBGRImage (see RSAPIWorkingDataSet) at any direction (e.g. coordinates $<$ 0 or $>$ width or height), use set the highest / lowest possible value (so concerning the example, 0 or width / height).

Parameters

cv::Point	∗upperLeft Pointer to the upperLeft coordinates of the box which shall be expanded. We use copies from the corresponding values of the RSAPIWorkingDataSet, because we do not want to expand the original, close-fitting boxes.
cv::Point	∗lowerRight Pointer to the lowerRight coordinates of the box which shall be expanded. We use copies from the corresponding values of the RSAPIWorkingDataSet, because we do not want to expand the original, close-fitting boxes.

References castedWorkingDataSet, expandPercentage, minExpandPixels, RoadSignAPI::RSAPIWorkingDataSet::originalImageHeight, and RoadSignAPI::RSAPIWorkingDataSet::originalImageWidth.

7.4.2.2 initByPipeSetup()

```
bool RoadSignAPI::DetectionBasedImageSlicer::initByPipeSetup ( ) [private], [virtual]
```

Initializes the filter.

As this filter does not need any specific environment variables, nothing really exciting is happenning here..

Returns

true, always

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References castedWorkingDataSet, FilterManagementLibrary::PipeSystem::PipeFilter::pipeWorkingDataSet, and FilterManagementLibrary::Logger::printfln().

7.4.2.3 process()

```
bool RoadSignAPI::DetectionBasedImageSlicer::process ( )  [private], [virtual]
```

The process function of this filter.

This function will be called by the ProcessingPipeline. Here the road signs detected by SSDRoadSignDetector will be cut out into smaller images (which only contain the sign), which will then be fed into the MobilenetV2RoadSignClassificator. To be fast, we assume that the coordinates calculated by the previous filter are correct!

Returns

true, always

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References castedWorkingDataSet, RoadSignAPI::RSAPIWorkingDataSet::cutOutImages, RoadSignAPI::RSAPIWorkingDataSet::detectedSigns, expandBox(), FilterManagementLibrary::PipeSystem::PipeFilter::invokeNext(), RoadSignAPI::DetectedSignDescriptor::lowerRight, RoadSignAPI::RSAPIWorkingDataSet::originalBGRImage, FilterManagementLibrary::PipeSystem::PipeFilter::pipeRegisteredFilters, and RoadSignAPI::DetectedSignDescriptor::upperLeft.

7.4.3 Member Data Documentation

7.4.3.1 castedWorkingDataSet

```
RSAPIWorkingDataSet* RoadSignAPI::DetectionBasedImageSlicer::castedWorkingDataSet  [private]
```

Just a pointer casted from PipeWorkingDataSet∗ to RSAPIWorkingDataSet∗, so we just don't have to do the casting every time we need it ;)

7.4.3.2 expandPercentage

```
float RoadSignAPI::DetectionBasedImageSlicer::expandPercentage = 0.1  [private]
```

This is the value we expand each box around a detected sign in any direction (x, -x, y, -x). Please refer to the class description for an explanation why we (have to) do this.

7.4.3.3 minExpandPixels

```
int RoadSignAPI::DetectionBasedImageSlicer::minExpandPixels = 5  [private]
```

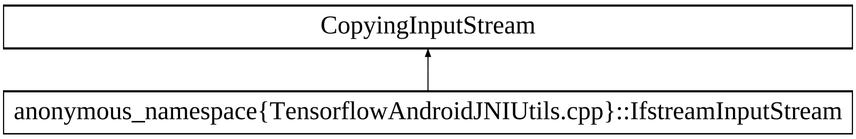
This is the minimum amount of pixels we expand each box around a detected sign in any direction (x, -x, y, -x). It is used to expand the box, if the value calculated using expandPercentage is to small: We want to expand AT LEAST 5 pixels (or more, respectively). Please refer to the class description for an explanation why we (have to) do this.

The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/DetectionBasedImageSlicer.h
- RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/DetectionBasedImageSlicer.cpp

7.5 anonymous_namespace{TensorflowAndroidJNIUtils.cpp}::IfstreamInputStream Class Reference

Inheritance diagram for anonymous_namespace{TensorflowAndroidJNIUtils.cpp}::IfstreamInputStream:


CopyingInputStream
anonymous_namespace{TensorflowAndroidJNIUtils.cpp}::IfstreamInputStream

Public Member Functions

- IfstreamInputStream (const std::string &file_name)
- ∼IfstreamInputStream ()
- int Read (void ∗buffer, int size)

Private Attributes

- std::ifstream ifs_

7.5.1 Constructor & Destructor Documentation

7.5.1.1 IfstreamInputStream()

```
anonymous_namespace{TensorflowAndroidJNIUtils.cpp}::IfstreamInputStream::IfstreamInputStream (
            const std::string & file_name )  [inline], [explicit]
```

7.5.1.2 ∼IfstreamInputStream()

```
anonymous_namespace{TensorflowAndroidJNIUtils.cpp}::IfstreamInputStream::∼IfstreamInputStream
( )  [inline]
```

7.5.2 Member Function Documentation

7.5.2.1 Read()

```
int anonymous_namespace{TensorflowAndroidJNIUtils.cpp}::IfstreamInputStream::Read (
            void ∗ buffer,
            int size )  [inline]
```

7.5.3 Member Data Documentation

7.5.3.1 ifs_

```
std::ifstream anonymous_namespace{TensorflowAndroidJNIUtils.cpp}::IfstreamInputStream::ifs_↩
[private]
```

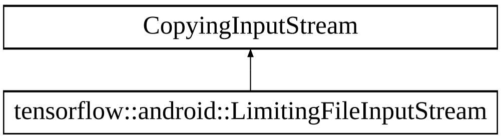
The documentation for this class was generated from the following file:

- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/TensorflowAndroidJNIUtils.cpp

7.6 tensorflow::android::LimitingFileInputStream Class Reference

```
#include <limiting_file_input_stream.h>
```

Inheritance diagram for tensorflow::android::LimitingFileInputStream:



Public Member Functions

- LimitingFileInputStream (int fd, int limit)
- ∼LimitingFileInputStream ()
- int Read (void ∗buffer, int size)
- int Skip (int count)

Private Attributes

- const int fd_
- int bytes_left_
- int errno_ = 0

7.6.1 Constructor & Destructor Documentation

7.6.1.1 LimitingFileInputStream()

```
tensorflow::android::LimitingFileInputStream::LimitingFileInputStream (
            int fd,
            int limit )  [inline]
```

7.6.1.2 ∼LimitingFileInputStream()

```
tensorflow::android::LimitingFileInputStream::∼LimitingFileInputStream ( )  [inline]
```

7.6.2 Member Function Documentation

7.6.2.1 Read()

```
int tensorflow::android::LimitingFileInputStream::Read (
            void * buffer,
            int size )  [inline]
```

References bytes_left_, errno_, and fd_.

7.6.2.2 Skip()

```
int tensorflow::android::LimitingFileInputStream::Skip (
            int count )  [inline]
```

References bytes_left_, and fd_.

7.6.3 Member Data Documentation

7.6.3.1 bytes_left_

```
int tensorflow::android::LimitingFileInputStream::bytes_left_  [private]
```

7.6.3.2 errno_

```
int tensorflow::android::LimitingFileInputStream::errno_ = 0  [private]
```

7.6.3.3 fd_

```
const int tensorflow::android::LimitingFileInputStream::fd_  [private]
```

The documentation for this class was generated from the following file:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/limiting_file_input_stream.h

7.7 FilterManagementLibrary::Logger Class Reference

Logger providing Desktop Linux <-> Android platform independent logging.

`#include <Logger.h>`

Static Public Member Functions

- static void printfln (const char *format,...)

  prints a new line to the console
- static void setLogTag (std::string logTag)

  sets the logTag of the logger

Static Private Member Functions

- static void getTimeString (std::string *timeStr)

  In-class helper function to get a string containing the current time.

Static Private Attributes

- static std::string logTag = "FilterManagementLibrary"

7.7.1 Detailed Description

Logger providing Desktop Linux <-> Android platform independent logging.

While compiling, it's checked if the library is built for Android or any other architecture. The logger class provides platform indepent logging by using preprocessor directives to only compile those functions that are needed on the given architecture. On Linux for example, the logger prints to stdout. On Android however, the logging library is used and the logger prints to the logcat output console.

7.7.2 Member Function Documentation

7.7.2.1 getTimeString()

```
void FilterManagementLibrary::Logger::getTimeString (
            std::string * timeStr )  [static], [private]
```

In-class helper function to get a string containing the current time.

Class intern helper function. It provides a string containing the current time as follows: hh:mm::ss hh being the hour mm being the minute ss being the second

Parameters

std::string	∗timeStr pointer to a string to wich the time string will be written to.

Returns

void.

7.7.2.2 printfln()

```
void FilterManagementLibrary::Logger::printfln (
            const char * format,
             ... )  [static]
```

prints a new line to the console

Takes a format c-string and a variable amount of parameters and formats an string accordingly. This string will be printed prefixed with the current time and the specified logTag.

Parameters

const	char∗ format c-string containing the string which shall be printed. May contain conversion specifiers like d, f etc.
...	variable of amount of parameters used to format the conversion specifiers.

References getTimeString(), and logTag.

7.7.2.3 setLogTag()

```
void FilterManagementLibrary::Logger::setLogTag (
            std::string logTag )  [static]
```

sets the logTag of the logger

The logTag is a prefix which will be printed when using printfln after the time string.

Parameters

std::string	logTag string containing the logTag

Returns

void

References logTag.

7.7.3 Member Data Documentation

7.7.3.1 logTag

```
std::string FilterManagementLibrary::Logger::logTag = "FilterManagementLibrary"  [static],
[private]
```

A tag to display in every print. By default it is set to "FilterManagementLibrary". With this, every use of printfln(...) would look like this: [00:38:12 - FilterManagementLibrary]: Logger print test. Notice that every print is prefixed with the current time for convenience.

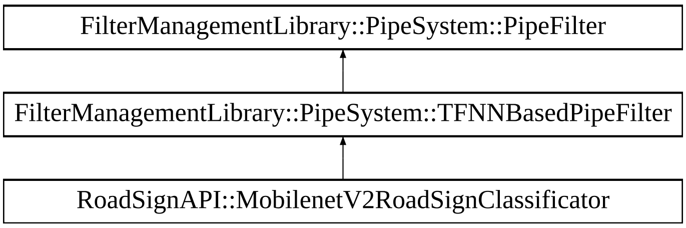
The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/Logger.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/Logger.cpp

7.8 RoadSignAPI::MobilenetV2RoadSignClassificator Class Reference

```
#include <MobilenetV2RoadSignClassificator.h>
```

Inheritance diagram for RoadSignAPI::MobilenetV2RoadSignClassificator:


FilterManagementLibrary::PipeSystem::PipeFilter
FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter
RoadSignAPI::MobilenetV2RoadSignClassificator

Public Member Functions

- MobilenetV2RoadSignClassificator (FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription, int numThreads, AAssetManager ∗const assetManager)

    Constructor of MobilenetV2RoadSignClassificator for Android environments. It basically just calls the TFNNBased←↩PipeFilter super constructor. Refer to it for a more detailed description. Under Android, the RoadSignAPI expects the model files of our neuronal networks to be placed in the assets folder of the app. Thus, this constructor needs to be passed a pointer to an AssetManager (from java), which is MANDATORY for the RoadSignAPI under Android!

- MobilenetV2RoadSignClassificator (FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription, int numThreads)

    Constructor of MobilenetV2RoadSignClassificator for non Android environments. It basically just calls the TFNN←↩BasedPipeFilter super constructor. Refer to it for a more detailed description.

Private Member Functions

- bool initByPipeSetup ()

  Initializes the filter.
- bool process ()

  Uses the neuronal network to classify the previous detected signs.
- void onNNEvaluationFinished (const FilterManagementLibrary::TFIntegration::TensorflowResultContainer resultContainer)

  Callback, will be called when the network finished it's prediction.
- void applyImageVectorFromOpenCVMat (cv::Mat ∗mat)

  Uses TensorflowOpenCVUtils to apply a mat as input to the network.
- bool isInUnwantedClasses (int classID) const

  Checks if the given class ID is in the list of unwanted ID's.

Private Attributes

- RSAPIWorkingDataSet ∗ castedWorkingDataSet
- int nnModelInputHeight
- int nnModelInputWidth
- float threshold = 0.95
- cv::Mat currentCutOutImage
- std::pair< float, int > currentRecognition
- bool validRecognition = false
- AAssetManager ∗const assetManager

Additional Inherited Members

7.8.1 Detailed Description

This filter is based on TFNNBasedPipeFilter and uses MobilenetV2 for (roadsign) classification.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 MobilenetV2RoadSignClassificator() [1/2]

```
RoadSignAPI::MobilenetV2RoadSignClassificator::MobilenetV2RoadSignClassificator (
             FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription nnModelDescription,
             int numThreads,
             AAssetManager *const assetManager )
```

Constructor of MobilenetV2RoadSignClassificator for Android environments. It basically just calls the TFNNBasedPipeFilter super constructor. Refer to it for a more detailed description. Under Android, the RoadSignAPI expects the model files of our neuronal networks to be placed in the assets folder of the app. Thus, this constructor needs to be passed a pointer to an AssetManager (from java), which is MANDATORY for the RoadSignAPI under Android!

All other class-member variables will be initialized to their default values.

Parameters

TensorflowNNModelDescription	nnModelDescription a description of the Tensorflow model which will be used by the filter.
int	numThreads number of threads Tensorflow is allowed to use for computation.
AAssetManager∗	const assetManager pointer to an Android Asset Manager which needs to be passed from the Java part of the Android App by any means. Manadatory to load the neuronal network models from the assets directory of the app.

7.8.2.2 MobilenetV2RoadSignClassificator() [2/2]

```
RoadSignAPI::MobilenetV2RoadSignClassificator::MobilenetV2RoadSignClassificator (
            FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription nnModel←↩
Description,
            int numThreads )
```

Constructor of MobilenetV2RoadSignClassificator for non Android environments. It basically just calls the TFNNBasedPipeFilter super constructor. Refer to it for a more detailed description.

All class-member variables will be initialized to their default values.

Parameters

TensorflowNNModelDescription	nnModelDescription a description of the Tensorflow model which will be used by the filter.
int	numThreads number of threads Tensorflow is allowed to use for computation.

7.8.3 Member Function Documentation

7.8.3.1 applyImageVectorFromOpenCVMat()

```
void RoadSignAPI::MobilenetV2RoadSignClassificator::applyImageVectorFromOpenCVMat (
            cv::Mat ∗ mat ) [private]
```

Uses TensorflowOpenCVUtils to apply a mat as input to the network.

Parameters

cv::Mat	∗mat pointer to an OpenCV Mat which shall be used as input.

References FilterManagementLibrary::TensorflowOpenCVUtils::fastApplyCVMatOnInputTensorFloat(), FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::getNNInputTensor(), and FilterManagementLibrary::←↩

PipeSystem::TFNNBasedPipeFilter::getNNModelDescription().

7.8.3.2 initByPipeSetup()

```
bool RoadSignAPI::MobilenetV2RoadSignClassificator::initByPipeSetup ( )  [private], [virtual]
```

Initializes the filter.

Will be called by ProcessingPipeline. Here, basically just the neuronal network model will be loaded. For this, the setupModelFromFile() or setupModelFromAssets() function will be called accordingly, depending whether we are under an Android environment or not. Refer to TFNNBasedPipeFilter for a more detailed description.

Returns

true, if the model could be loaded successfully, false otherwise

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References assetManager, castedWorkingDataSet, currentCutOutImage, FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::getNNModelDescription(), FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputHeight, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputWidth, nnModelInputHeight, nnModelInputWidth, FilterManagementLibrary::PipeSystem::PipeFilter::pipeWorkingDataSet, FilterManagementLibrary::Logger::printfln(), FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::setupModelFromAssets(), and FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::setupModelFromFile().

7.8.3.3 isInUnwantedClasses()

```
bool RoadSignAPI::MobilenetV2RoadSignClassificator::isInUnwantedClasses (
            int classID ) const  [private]
```

Checks if the given class ID is in the list of unwanted ID's.

Parameters

int	classID ID of the class which shall be checked.

Returns

true if the given ID is in the list of unwanted class ID's (i.e. misc classes), false otherwise.

References onNNEvaluationFinished().

7.8.3.4 onNNEvaluationFinished()

```
void RoadSignAPI::MobilenetV2RoadSignClassificator::onNNEvaluationFinished (
            const FilterManagementLibrary::TFIntegration::TensorflowResultContainer result↩
Container )  [private], [virtual]
```

Callback, will be called when the network finished it's prediction.

In process(), the TFNNBasedPipeFilter's (super class) evaluteInputVectorByNN() will be called. If it was successfull, this callback will be called and passed a TensorflowResultContainer with the result of the inference of the neuronal network model. Here we will check the prediction and find the highest confidence prediction, which exceeds a certain threshold. The prediction itself will be examined in process() itself (remember, this is a sync function! It will be called immediately after evaluteInputVectorByNN() was called and then will return to process() again).

Parameters

const	TensorflowResultContainer resultContainer contains the results (in Tensorflow tensors) of the network prediction.

Implements FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter.

References applyImageVectorFromOpenCVMat(), currentRecognition, i, threshold, and validRecognition.

7.8.3.5 process()

```
bool RoadSignAPI::MobilenetV2RoadSignClassificator::process ( )  [private], [virtual]
```

Uses the neuronal network to classify the previous detected signs.

Iterates over all the images the DetectionBasedImageSlicer generated from the originalBGRImage and uses the neuronal network model to classify them (uses evaluateInputVectorByNN() of TFNNBasedPipeFilter super class. Will set the class ID and the confidence accordingly. All classes which are unwanted (i.e. misc classes) will be filtered out, in other words they won't be added to classifierApprovedSigns of RSAPIWorkingDataSet's.

Returns

true if evaluateInputVectorByNN() return true, false otherwise (does NOT return false if no signs could be classified!)

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References applyImageVectorFromOpenCVMat(), castedWorkingDataSet, RoadSignAPI::RSAPIWorkingData↩Set::classifierApprovedSigns, currentCutOutImage, currentRecognition, RoadSignAPI::RSAPIWorkingData↩Set::cutOutImages, FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::evaluateInputVectorByNN(), i, FilterManagementLibrary::PipeSystem::PipeFilter::indicateProcessingFinished(), FilterManagementLibrary::↩PipeSystem::PipeFilter::invokeNext(), isInUnwantedClasses(), FilterManagementLibrary::PipeSystem::PipeFilter↩::pipeRegisteredFilters, and validRecognition.

7.8.4 Member Data Documentation

7.8.4.1 assetManager

```
AAssetManager* const RoadSignAPI::MobilenetV2RoadSignClassificator::assetManager [private]
```

A pointer to an AssetManager which can be passed via the constructor. We need it to be able to load model files from the Android assets folder, which we want to do so the files do not have to be placed in the normal user space.

7.8.4.2 castedWorkingDataSet

```
RSAPIWorkingDataSet* RoadSignAPI::MobilenetV2RoadSignClassificator::castedWorkingDataSet [private]
```

Just a pointer casted from PipeWorkingDataSet* to RSAPIWorkingDataSet*, so we just don't have to do the casting every time we need it ;)

7.8.4.3 currentCutOutImage

```
cv::Mat RoadSignAPI::MobilenetV2RoadSignClassificator::currentCutOutImage [private]
```

A cv::Mat were we will temporarily store a resized image matrix while we iterate over {

See also

RSAPIWorkingDataSet}::cutOutImages (we need to resize them to the size our network model expects).

7.8.4.4 currentRecognition

```
std::pair<float, int> RoadSignAPI::MobilenetV2RoadSignClassificator::currentRecognition [private]
```

Inter-class copy of top rated recognition we determined in on onNNEvaluationFinished(...). You may refer to the process() function of this class for a description of where this is used.

7.8.4.5 nnModelInputHeight

```
int RoadSignAPI::MobilenetV2RoadSignClassificator::nnModelInputHeight [private]
```

Just a copy from the input height provided in the TensorflowNNModelDescription of the underlaying TensorflowN↩NInstance.

7.8.4.6 nnModelInputWidth

```
int RoadSignAPI::MobilenetV2RoadSignClassificator::nnModelInputWidth [private]
```

Just a copy from the input height provided in the TensorflowNNModelDescription of the underlying TensorflowNN↩Instance.

7.8.4.7 threshold

```
float RoadSignAPI::MobilenetV2RoadSignClassificator::threshold = 0.95  [private]
```

Threshold for the output confidence of the neuronal network model. Means a result needs to achieve a confidence >= threshold to be candidate for list of top N predictions.

7.8.4.8 validRecognition

```
bool RoadSignAPI::MobilenetV2RoadSignClassificator::validRecognition = false  [private]
```

If none of the outputs of the network beats the threshold, we currentRecognition may store the result of a previous classification, which would lead to wrong results. This variable states if there was a valid recognition in the last evaluateImageByNN() call.

The documentation for this class was generated from the following files:

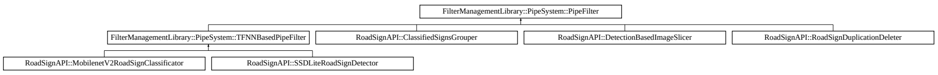
- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/MobilenetV2RoadSignClassificator.↩ h
- RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/MobilenetV2RoadSignClassificator.↩ cpp

7.9 FilterManagementLibrary::PipeSystem::PipeFilter Class Reference

Base class were all filters are derived from.

```
#include <PipeFilter.h>
```

Inheritance diagram for FilterManagementLibrary::PipeSystem::PipeFilter:


FilterManagementLibrary::PipeSystem::PipeFilter
FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter
RoadSignAPI::ClassifiedSignsGrouper
RoadSignAPI::DetectionBasedImageSlicer
RoadSignAPI::RoadSignDuplicationDeleter
RoadSignAPI::MobilenetV2RoadSignClassificator
RoadSignAPI::SSDLiteRoadSignDetector

Public Member Functions

- PipeFilter ()

  Standard constructor of PipeFilter.
- int getFilterID () const

  Returns the ID of the filter.
- int getNextDesiredFilter () const

  Returns the ID of the filter which shall be invoked next.
- bool hasMarkedNextDesiredFilter () const

  Returns whether a filter which shall be invoked next has been marked.
- bool hasMarkedProcessingFinished () const

  Returns whether the processing on the current DataSet is finished.
- virtual ∼PipeFilter ()

  Standard destructor of PipeFilter. Has no functionality yet.

Protected Member Functions

- virtual bool initByPipeSetup ()=0
- virtual bool process ()=0
- void indicateProcessingFinished ()

  Function for derived filters to indicate all processing is done.
- void invokeNext (const int filterID)
- void reset ()

  Function to reset basic variables of the filter, called by the pipe.

Protected Attributes

- PipeWorkingDataSet ∗ pipeWorkingDataSet
- PipeRegisteredFilters ∗ pipeRegisteredFilters

Private Member Functions

- void setCredentials (int filterID, PipeRegisteredFilters ∗pipeRegisteredFilters, PipeWorkingDataSet ∗pipe↩ WorkingDataSet)

  Function which is used by ProcessingPipeline to hand in some variables.

Private Attributes

- int filterID
- int nextDesiredFilter = -1
- bool markedNextDesiredFilter = false
- bool markedProcessingFinished = false

Friends

- class ProcessingPipeline

7.9.1 Detailed Description

Base class were all filters are derived from.

A filter in terms of the pipes and filters architecture is a single processing step. Originally, each filter has a data input and a data output. In our case, however, the data input and output is mapped to a single object (the data object, see PipeWorkingDataSet) allocated, on the heap, on which all operations are performed. In each processing step, the filter transforms the specific part of the data object that affects him. For this, each filter implements a function process() which defines the (conversion) functionality of the filter.

During each procession, parts of the data object may be added, edited or removed completely. The type of conversion is determined by each filter. As each filter knows the structure of the data object at compile time, compatibility between each filters of a pipe is guaranteed! This means, each filter get's at least those variables of the data object it expects.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 PipeFilter()

```
FilterManagementLibrary::PipeSystem::PipeFilter::PipeFilter ( )
```

Standard constructor of PipeFilter.

Initialises all class member variables to their default values.

7.9.2.2 ∼PipeFilter()

```
FilterManagementLibrary::PipeSystem::PipeFilter::∼PipeFilter ( )  [virtual]
```

Standard destructor of PipeFilter. Has no functionality yet.

7.9.3 Member Function Documentation

7.9.3.1 getFilterID()

```
int FilterManagementLibrary::PipeSystem::PipeFilter::getFilterID ( ) const
```

Returns the ID of the filter.

Returns

int filterID the ID of the filter.

References filterID, and getNextDesiredFilter().

7.9.3.2 getNextDesiredFilter()

```
int FilterManagementLibrary::PipeSystem::PipeFilter::getNextDesiredFilter ( ) const
```

Returns the ID of the filter which shall be invoked next.

Whenever a Filter returns true in it's process() function, it needs to inform the ProcessingPipeline about what to do next (is the processing finished, or is there a next filter which shall be invoked ?). For the latter, the ID of the next filter will be set by invokeNextFilter(...) and can be derived by the ProcessingPipeline using this getter-function.

Returns

int ID of the filter which shall be invoked next.

References hasMarkedNextDesiredFilter(), and nextDesiredFilter.

7.9.3.3 hasMarkedNextDesiredFilter()

```
bool FilterManagementLibrary::PipeSystem::PipeFilter::hasMarkedNextDesiredFilter ( ) const
```

Returns whether a filter which shall be invoked next has been marked.

Whenever a Filter returns true in it's process() function, it needs to inform the ProcessingPipeline about what to do next (is the processing finished, or is there a next filter which shall be invoked ?). For the latter, the ProcessingPipe can check if a next filter is set using this function.

Returns

bool true if a next filter has been marked, else otherwise

References hasMarkedProcessingFinished(), and markedNextDesiredFilter.

7.9.3.4 hasMarkedProcessingFinished()

```
bool FilterManagementLibrary::PipeSystem::PipeFilter::hasMarkedProcessingFinished ( ) const
```

Returns whether the processing on the current DataSet is finished.

Whenever a Filter returns true in it's process() function, it needs to inform the ProcessingPipeline about what to do next (is the processing finished, or is there a next filter which shall be invoked ?). Regarding the first, the ProcessingPipeline can check that using this function.

Returns

bool true if the processing is marked finished, else otherwise

References markedProcessingFinished.

7.9.3.5 indicateProcessingFinished()

```
void FilterManagementLibrary::PipeSystem::PipeFilter::indicateProcessingFinished ( ) [protected]
```

Function for derived filters to indicate all processing is done.

Whenever a Filter returns true in it's process() function, it needs to inform the ProcessingPipeline about what to do next (e.g. which filter needs to be invoked next). However if a Filter decides that all processing is finsished and no following filter shall do further calculations on / modifications to the PipeWorkingDataSet, the filter shall indicate that the processing is finished (successfully, because it would return true !)

Returns

void

References invokeNext(), and markedProcessingFinished.

7.9.3.6 initByPipeSetup()

```
virtual bool FilterManagementLibrary::PipeSystem::PipeFilter::initByPipeSetup ( )  [protected],
[pure virtual]
```

Get's called when the pipe's setup function is called after all the filters have been registered.

Implemented in RoadSignAPI::MobilenetV2RoadSignClassificator, RoadSignAPI::DetectionBasedImageSlicer, RoadSignAPI::SSDLiteRoadSignDetector, RoadSignAPI::RoadSignDuplicationDeleter, and RoadSignAPI::ClassifiedSignsGrouper.

7.9.3.7 invokeNext()

```
void FilterManagementLibrary::PipeSystem::PipeFilter::invokeNext (
            const int filterID )  [protected]
```

References filterID, markedNextDesiredFilter, and nextDesiredFilter.

7.9.3.8 process()

```
virtual bool FilterManagementLibrary::PipeSystem::PipeFilter::process ( )  [protected], [pure
virtual]
```

Implemented in RoadSignAPI::MobilenetV2RoadSignClassificator, RoadSignAPI::DetectionBasedImageSlicer, RoadSignAPI::SSDLiteRoadSignDetector, RoadSignAPI::RoadSignDuplicationDeleter, and RoadSignAPI::ClassifiedSignsGrouper.

7.9.3.9 reset()

```
void FilterManagementLibrary::PipeSystem::PipeFilter::reset ( )  [protected]
```

Function to reset basic variables of the filter, called by the pipe.

This function will be called by the pipe on the beginning of the processCurrentDataSet() function. It is intended to only reset thoe variables used for inter-pipe-information flow, so it only affects whether the processing is marked finished and the filter which is marked next. Reset of filter-specific variables will be done in the Filter's process function. This is NO virtual function and won't be inherited!

Returns

void

References markedNextDesiredFilter, markedProcessingFinished, and nextDesiredFilter.

7.9.3.10 setCredentials()

```
void FilterManagementLibrary::PipeSystem::PipeFilter::setCredentials (
            int filterID,
            PipeRegisteredFilters * pipeRegisteredFilters,
            PipeWorkingDataSet * pipeWorkingDataSet )  [private]
```

Function which is used by ProcessingPipeline to hand in some variables.

When a filter is registered to a Pipeline, it needs to know on which objects the pipe is working (PipeWorkingDataSet) and which filters are registered to it (PipeRegisteredFilters). This is done through this function, which is called by ProcessingPipeline in it's registerFilter(...) function. It is not intended that this function is called anywhere outside of ProcessingPipeline.

Parameters

int	filterID will be the ID used to identify the filter
PipeRegisteredFilters∗	pipeRegisteredFilters pointer to the struct containing the ID's of all filters registed to the pipe
PipeWorkingDataSet∗	pipeWorkingDataSet pointer to the dataSet the pipe is working on

Returns

void

References filterID, indicateProcessingFinished(), pipeRegisteredFilters, and pipeWorkingDataSet.

7.9.4 Friends And Related Function Documentation

7.9.4.1 ProcessingPipeline

```
friend class ProcessingPipeline  [friend]
```

7.9.5 Member Data Documentation

7.9.5.1 filterID

```
int FilterManagementLibrary::PipeSystem::PipeFilter::filterID  [private]
```

7.9.5.2 markedNextDesiredFilter

```
bool FilterManagementLibrary::PipeSystem::PipeFilter::markedNextDesiredFilter = false  [private]
```

Set to true when the invokeNext() function has been called. A getter function is provided so that the ProcessingPipeline can query if a next filter has been marked.

7.9.5.3 markedProcessingFinished

```
bool FilterManagementLibrary::PipeSystem::PipeFilter::markedProcessingFinished = false  [private]
```

This is a hint for the ProcessingPipeline; any filter can indicate that the processing of the pipeWorkingDataSet has is finished and no more other filters of the pipe shall be called. A getter function is provided so that the ProcessingPipeline can query if a next filter has been marked.

7.9.5.4 nextDesiredFilter

```
int FilterManagementLibrary::PipeSystem::PipeFilter::nextDesiredFilter = -1  [private]
```

This is a hint for the ProcessingPipeline; any filter can indicate which filter should be invoked next. For this, PipeFilter provides invokeNext() function, whose result will be stored in this variable. A getter function is provided so that the ProcessingPipeline can query the ID of the next desired filter (if any).

7.9.5.5 pipeRegisteredFilters

```
PipeRegisteredFilters* FilterManagementLibrary::PipeSystem::PipeFilter::pipeRegisteredFilters
[protected]
```

Pointer to a derivate of PipeRegisteredFilters, being a container for all ID's of the filters registered to the pipe, mapped to a unique name. Via this all filters know the ID's of every other filter registered to the pipe. The pointer will be passed to all filters using their setCredentials(...) function.

7.9.5.6 pipeWorkingDataSet

```
PipeWorkingDataSet* FilterManagementLibrary::PipeSystem::PipeFilter::pipeWorkingDataSet  [protected]
```

Pointer to a derivate of PipeWorkingDataSet, being the dataSet the filters of the pipe will work on. Will be passed to all filters using their setCredentials(...) function.

The documentation for this class was generated from the following files:

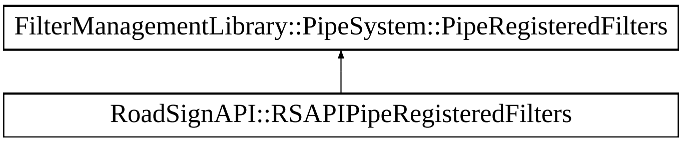
- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/PipeSystem/PipeFilter.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/PipeSystem/PipeFilter.cpp

7.10 FilterManagementLibrary::PipeSystem::PipeRegisteredFilters Struct Reference

Template struct.

```
#include <PipeRegisteredFilters.h>
```

Inheritance diagram for FilterManagementLibrary::PipeSystem::PipeRegisteredFilters:


FilterManagementLibrary::PipeSystem::PipeRegisteredFilters
RoadSignAPI::RSAPIPipeRegisteredFilters

7.10.1 Detailed Description

Template struct.

In from this derived structs, the user should organize it's Filters. We provide this to be able store the ID's of the registered Filters and get them from within any Filter which belongs to the corresponding pipe. Refer to Processing↩ Pipeline for more context; when it's used and how.

The documentation for this struct was generated from the following file:

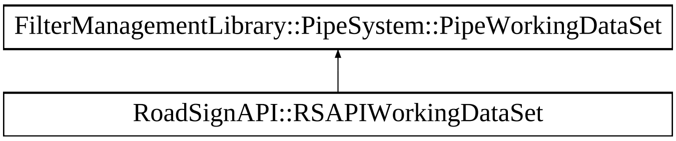
- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/PipeSystem/PipeRegistered↩ Filters.h

7.11 FilterManagementLibrary::PipeSystem::PipeWorkingDataSet Struct Reference

Template structfor PipeWorkingDataSets.

```
#include <PipeWorkingDataSet.h>
```

Inheritance diagram for FilterManagementLibrary::PipeSystem::PipeWorkingDataSet:


FilterManagementLibrary::PipeSystem::PipeWorkingDataSet
RoadSignAPI::RSAPIWorkingDataSet

7.11.1 Detailed Description

Template structfor PipeWorkingDataSets.

In from this derived structs, the working data set for the specific pipe environment should be implemented. Refer to ProcessingPipeline for more context; when it's used and how.

The documentation for this struct was generated from the following file:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/PipeSystem/PipeWorking↩ DataSet.h

7.12 FilterManagementLibrary::PipeSystem::ProcessingPipeline Class Reference

Pipeline of the pipes and filters architecture.

```
#include <ProcessingPipeline.h>
```

Public Types

- enum ErrorType {
  ERROR_NONE, ERROR_LOGIC_FAULT, ERROR_FILTER_INDICATED_FAILURE, ERROR_FILTER_ID_NOT_FOUND,
  ERROR_FILTER_SETUP_FAILED }

Public Member Functions

- ProcessingPipeline (PipeWorkingDataSet ∗workingDataSet, PipeRegisteredFilters ∗pipeRegisteredFiltersHeader, bool manageExternalAllocatedRessources=true)

  Standard constructor of ProcessingPipeline.
- void registerFilter (PipeFilter ∗filter, int ∗filterID)

  Registers a filter onto the ProcessingPipeline.
- ErrorType getLastError () const

  Returns an (enum) ID of the last error that happened.
- bool setup ()

  Set up the pipe and it's filters.
- bool processCurrentDataSet ()

  Process the PipeWorkingDataSet.
- ∼ProcessingPipeline ()

  Destructor of ProcessingPipeline.

Protected Attributes

- PipeWorkingDataSet ∗ workingDataSet
- PipeRegisteredFilters ∗ pipeRegisteredFiltersHeader

Private Attributes

- int currentFilterID = 0
- bool processingFinishied = false
- bool manageExternalAllocatedRessources = true
- std::vector< PipeFilter ∗ > registeredFilters
- ErrorType lastError

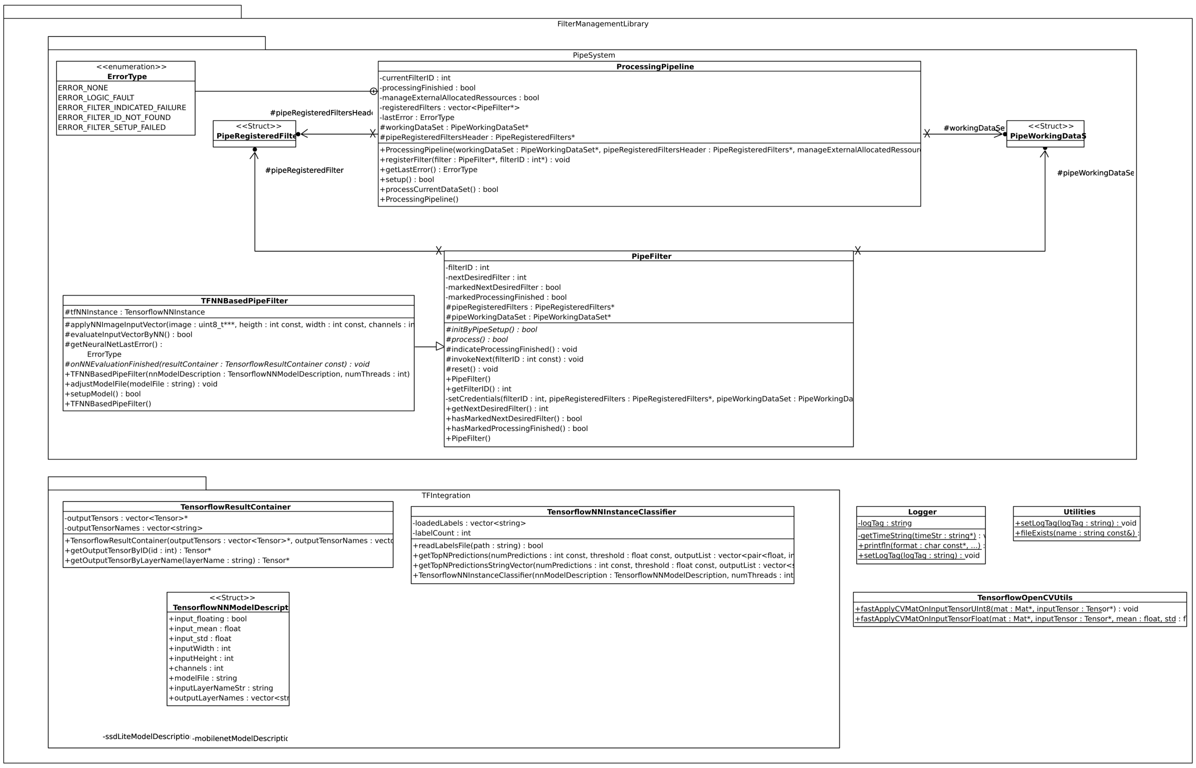
7.12.1 Detailed Description

Pipeline of the pipes and filters architecture.

The Pipes-and-Filters-Architecture is a software architectural pattern usually used for systems processing data or data streams in a specific way. A pipe, which can be seen as a sequence of processing operations, represents the connection between the individual filters (processing steps). You may refer to PipeFilter for a more detailed explanation of the filters part of the architecture.

The output of a processing step (filter) can be seen as the input for the subsequent processing step, although in this implementation the input and output data is mapped to a single object allocated on the heap, the PipeWorkingDataSet.

As this implementation is based on the "Tee-and-Joins-Pipes-and-Filters-Architecture", the order in which the filters are executed can change dynamically. For now this is implemented as follows: Each filter provides a function process(). In this function a filter either has to indicate which filter shall be invoked next, or that it claims to be the last filter of the processing at least for the current iteration, which means the current data set has been fully processed. The filter may decide differently for other input data, however. Strictly speaking, this means every filter has to give a hint to the pipe to allow it to decide what to do next. For this, each filter knows the ID's of all registered filters on the pipe itself does belong to. These ID's are provided via a shared object, a derivate of the PipeRegisteredFilters struct.


FilterManagementLibrary
PipeSystem
<<enumeration>>
ErrorType
ERROR_NONE
ERROR_LOGIC_FAULT
ERROR_FILTER_INDICATED_FAILURE
ERROR_FILTER_ID_NOT_FOUND
ERROR_FILTER_SETUP_FAILED
#pipeRegisteredFiltersHeade
<<Struct>>
PipeRegisteredFilte
#pipeRegisteredFilter
ProcessingPipeline
-currentFilterID : int
-processingFinishied : bool
-manageExternalAllocatedRessources : bool
-registeredFilters : vector<PipeFilter*>
-lastError : ErrorType
#workingDataSet : PipeWorkingDataSet*
#pipeRegisteredFiltersHeader : PipeRegisteredFilters*
+ProcessingPipeline(workingDataSet : PipeWorkingDataSet*, pipeRegisteredFiltersHeader : PipeRegisteredFilters*, manageExternalAllocatedRessour
+registerFilter(filter : PipeFilter*, filterID : int*) : void
+getLastError() : ErrorType
+setup() : bool
+processCurrentDataSet() : bool
+ProcessingPipeline()
#workingDataSe
<<Struct>>
PipeWorkingDataS
#pipeWorkingDataSe
TFNNBasedPipeFilter
#tfNNInstance : TensorflowNNInstance
#applyNNImageInputVector(image : uint8_t***, heigth : int const, width : int const, channels : in
#evaluateInputVectorByNN() : bool
#getNeuralNetLastError() :
ErrorType
#onNNEvaluationFinished(resultContainer : TensorflowResultContainer const) : void
+TFNNBasedPipeFilter(nnModelDescription : TensorflowNNModelDescription, numThreads : int)
+adjustModelFile(modelFile : string) : void
+setupModel() : bool
+TFNNBasedPipeFilter()
PipeFilter
-filterID : int
-nextDesiredFilter : int
-markedNextDesiredFilter : bool
-markedProcessingFinished : bool
#pipeRegisteredFilters : PipeRegisteredFilters*
#pipeWorkingDataSet : PipeWorkingDataSet*
#initByPipeSetup() : bool
#process() : bool
#indicateProcessingFinished() : void
#invokeNext(filterID : int const) : void
#reset() : void
+PipeFilter()
+getFilterID() : int
-setCredentials(filterID : int, pipeRegisteredFilters : PipeRegisteredFilters*, pipeWorkingDataSet : PipeWorkingDa
+getNextDesiredFilter() : int
+hasMarkedNextDesiredFilter() : bool
+hasMarkedProcessingFinished() : bool
+PipeFilter()
TFIntegration
TensorflowResultContainer
-outputTensors : vector<Tensor>*
-outputTensorNames : vector<string>
+TensorflowResultContainer(outputTensors : vector<Tensor>*, outputTensorNames : vectc
+getOutputTensorByID(id : int) : Tensor*
+getOutputTensorByLayerName(layerName : string) : Tensor*
TensorflowNNInstanceClassifier
-loadedLabels : vector<string>
-labelCount : int
+readLabelsFile(path : string) : bool
+getTopNPredictions(numPredictions : int const, threshold : float const, outputList : vector<pair<float, ir
+getTopNPredictionsStringVector(numPredictions : int const, threshold : float const, outputList : vector<s
+TensorflowNNInstanceClassifier(nnModelDescription : TensorflowNNModelDescription, numThreads : int
<<Struct>>
TensorflowNNModelDescript
+input_floating : bool
+input_mean : float
+input_std : float
+inputWidth : int
+inputHeight : int
+channels : int
+modelFile : string
+inputLayerNameStr : string
+outputLayerNames : vector<str
-ssdLiteModelDescriptio
-mobilenetModelDescriptio
Logger
-logTag : string
-getTimeString(timeStr : string*) : v
+printfln(format : char const*, ...) :
+setLogTag(logTag : string) : void
Utilities
+setLogTag(logTag : string) : void
+fileExists(name : string const&) :
TensorflowOpenCVUtils
+fastApplyCVMatOnInputTensorUInt8(mat : Mat*, inputTensor : Tensor*) : void
+fastApplyCVMatOnInputTensorFloat(mat : Mat*, inputTensor : Tensor*, mean : float, std : f

7.12.2 Member Enumeration Documentation

7.12.2.1 ErrorType

```
enum FilterManagementLibrary::PipeSystem::ProcessingPipeline::ErrorType
```

Enumerator

ERROR_NONE	Standard value set on initialization. Needed because getLastError() shall not return a random value (which could cause unexpected behavior)
ERROR_LOGIC_FAULT	E.g. if a filter neither marked a next filter nor indicated the workingDataSet has been fully processed.
ERROR_FILTER_INDICATED_FAILURE	If a filter returned false in process()
ERROR_FILTER_ID_NOT_FOUND	If a filter wants to invoke a next filter with an ID that is not assigned.
ERROR_FILTER_SETUP_FAILED	When we reun initByPipeSetup() on a filter and it returned false, which means the filter could not be set up and all filters depending on it may not be able to perform their tasks.

7.12.3 Constructor & Destructor Documentation

7.12.3.1 ProcessingPipeline()

```
FilterManagementLibrary::PipeSystem::ProcessingPipeline::ProcessingPipeline (
            PipeWorkingDataSet * workingDataSet,
            PipeRegisteredFilters * pipeRegisteredFiltersHeader,
            bool manageExternallyAllocatedRessources = true )
```

Standard constructor of ProcessingPipeline.

The ProcessingPipeline needs to know the PipeWorkingDataSet on which all operations of the filters will be performed and the PipeRegisteredFilters struct which will contain the ID's of the filters registered to the pipe. Pointers to these coressponding objects need to be passed to this constructor. They will be passed to all registered filters. Also via manageExternalAllocatedRessources, it can be determined whether the ProcessingPipeline will "take ownership" of all those variables allocated on the heap (PipeWorkingDataSet, PipeRegisteredFilters and all registered Filters themselves), which means it will delete them on destruction. All other class members will be initialised to their defult values.

Parameters

*PipeWorkingDataSet**	a pointer to an object of a struct derived from PipeWorkingDataSet. This is the data set the filters work on and will be passed to all registered filters.
*PipeRegisteredFilters**	a pointer to an object of a struct derived from PipeRegisteredFilters. The struct should contain (integer) values to the ID's of all registered Filters. The filters need this for their invokeNext(const int filterID) function to know the ID of the filter they want to invoke next.
bool	manageExternalAllocatedRessources indicate whether the ProcessingPipeline shall take ownership of all variables passed to it which were allocated on the heap, therefore deleting them on destruction.

References registerFilter().

7.12.3.2 ∼ProcessingPipeline()

```
FilterManagementLibrary::PipeSystem::ProcessingPipeline::∼ProcessingPipeline ( )
```

Destructor of ProcessingPipeline.

If manageExternalAllocatedRessources was set to true in constructor, all variables on the heap (PipeWorkingDataSet, PipeRegisteredFilters, the filters itselves) will be deleted. Otherwise the destructor does do nothing.

Returns

ProcessingPipeline::ErrorType the enum value of the last error that happened.

References manageExternalAllocatedRessources, pipeRegisteredFiltersHeader, registeredFilters, and workingDataSet.

7.12.4 Member Function Documentation

7.12.4.1 getLastError()

```
FilterManagementLibrary::PipeSystem::ProcessingPipeline::ErrorType FilterManagementLibrary::↩
PipeSystem::ProcessingPipeline::getLastError ( ) const
```

Returns an (enum) ID of the last error that happened.

Returns

ProcessingPipeline::ErrorType the enum value of the last error that happened.

References lastError.

7.12.4.2 processCurrentDataSet()

```
bool FilterManagementLibrary::PipeSystem::ProcessingPipeline::processCurrentDataSet ( )
```

Process the PipeWorkingDataSet.

Will iterate over all filters and call their process() functions, which means the DataSet will be processed. Each filter will (should !) indicate either if the processing has been finished, or if (and which) a filter shall be invoked next, as long as their process function returns true. Otherwise, if a filter returns false, this means the current dataset couldn't be processed. Either way, in any case of failure, lastError will be set accordingly.

Returns

bool true if all filters return true in their process() functions AND either indicate which filter to invoke next or that the processing is finished, false otherwise.

References currentFilterID, FilterManagementLibrary::PipeSystem::PipeFilter::getNextDesiredFilter(), FilterManagementLibrary::PipeSystem::PipeFilter::hasMarkedNextDesiredFilter(), FilterManagementLibrary::PipeSystem::PipeFilter::hasMarkedProcessingFinished(), lastError, FilterManagementLibrary::Logger::printfln(), FilterManagementLibrary::PipeSystem::PipeFilter::process(), processingFinishied, registeredFilters, and FilterManagementLibrary::PipeSystem::PipeFilter::reset().

7.12.4.3 registerFilter()

```
void FilterManagementLibrary::PipeSystem::ProcessingPipeline::registerFilter (
            PipeFilter * filter,
            int * filterID )
```

Registers a filter onto the ProcessingPipeline.

Each filter which shall be included into the ProcessingPipeline needs to be registered to it using this function.

Parameters

PipeFilter∗	filter pointer to the filter to register
int	filterID a pointer to an integer whose value will be changed in this function. It will contain the ID of the newly registered filter afterwards.

Returns

void

References pipeRegisteredFiltersHeader, registeredFilters, FilterManagementLibrary::PipeSystem::PipeFilter↩ ::setCredentials(), and workingDataSet.

7.12.4.4 setup()

```
bool FilterManagementLibrary::PipeSystem::ProcessingPipeline::setup ( )
```

Set up the pipe and it's filters.

oracion Will iterate over all filters and call initByPipeSetup() on them. On failure, lastError will be set accordingly.

Returns

bool true if all filters return true in their initByPipeSetup() functions, false otherwise.

References FilterManagementLibrary::PipeSystem::PipeFilter::getFilterID(), FilterManagementLibrary::Pipe↩ System::PipeFilter::initByPipeSetup(), lastError, FilterManagementLibrary::Logger::printfln(), processCurrent↩ DataSet(), and registeredFilters.

7.12.5 Member Data Documentation

7.12.5.1 currentFilterID

```
int FilterManagementLibrary::PipeSystem::ProcessingPipeline::currentFilterID = 0  [private]
```

ID of the currently active filter. Each filter ID is unique and has to be positive. Uniqueness is assured by the pipe itself due to continuous numerating in registerFilter()

7.12.5.2 lastError

```
ErrorType FilterManagementLibrary::PipeSystem::ProcessingPipeline::lastError  [private]
```

Error code of the last error that happened. See ProcessingPipeline::ErrorType

7.12.5.3 manageExternalAllocatedRessources

```
bool FilterManagementLibrary::PipeSystem::ProcessingPipeline::manageExternalAllocatedRessources
= true  [private]
```

Indicate whether we shall manage the heap allocated memory of workingDataSet, pipeRegisteredFiltersHeader and the themselves, i.e. explicitly delete those externally allocated objects in our destructor.

7.12.5.4 pipeRegisteredFiltersHeader

```
PipeRegisteredFilters* FilterManagementLibrary::PipeSystem::ProcessingPipeline::pipeRegistered←
FiltersHeader  [protected]
```

Pointer to a derivate of PipeRegisteredFilters, being a container for all ID's of the filters registered to the pipe, mapped to a unique name. Via this all filters know the ID's of every other filter registered to the pipe. The pointer will be passed to all filters using their setCredentials(...) function.

7.12.5.5 processingFinishied

```
bool FilterManagementLibrary::PipeSystem::ProcessingPipeline::processingFinishied = false
[private]
```

Stores whether the last filter indicated the current data set has been fully processed.

7.12.5.6 registeredFilters

```
std::vector<PipeFilter*> FilterManagementLibrary::PipeSystem::ProcessingPipeline::registered←
Filters  [private]
```

Just a list containing pointers to all filters registered on this instance of the pipe.

7.12.5.7 workingDataSet

```
PipeWorkingDataSet* FilterManagementLibrary::PipeSystem::ProcessingPipeline::workingDataSet
[protected]
```

Pointer to a derivate of PipeWorkingDataSet, being the dataSet the filters of the pipe will work on. Will be passed to all filters using their setCredentials(...) function.

The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/PipeSystem/Processing←Pipeline.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/PipeSystem/Processing←Pipeline.cpp

7.13 RoadSignAPI::RoadSignAPI Class Reference

```
#include <RoadSignAPI.h>
```

Public Member Functions

- bool init ()

  Initializes the RoadSignAPI's static interface.
- bool feedImage (cv::Mat iamge)

  Takes an OpenCV Mat uses the filter to examine it for road signs.
- const std::vector< DetectedSignDescriptor > ∗ getDetectedSigns ()

  Returns a vector of all detected (not classified!) signs.
- void getClassifierApprovedDetectedSigns (std::vector< const DetectedSignDescriptor ∗> ∗detectedSigns)

  Returns a vector of all classified signs.
- const std::vector< DetectedSignCombination > ∗const getDetectedSignCombinations () const

  Returns a vector of all DetectedSignCombination}s.
- RoadSignAPI (FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription detectorModel↩ Description, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription classificatorModel↩ Description, const int numThreads, AAssetManager ∗const assetManager)

  Constructor of the RoadSignAPI for Android environments.
- RoadSignAPI (FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription detectorModel↩ Description, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription classificatorModel↩ Description, const int numThreads)

  Constructor of the RoadSignAPI for non Android environments.

Static Public Member Functions

- static bool staticInit (const int numThreads, AAssetManager ∗const assetManager)

  Initializes the RoadSignAPI's static interface.
- static bool staticInit (const int numThreads)

  Initializes the RoadSignAPI's static interface.
- static bool staticFeedImage (cv::Mat image)

  Takes an OpenCV Mat uses the filter to examine it for road signs.
- static const std::vector< DetectedSignDescriptor > ∗ staticGetDetectedSigns ()

  Returns a vector of all detected (not classified!) signs.
- static void staticGetClassifierApprovedDetectedSigns (std::vector< const DetectedSignDescriptor ∗> ∗detectedSigns)

  Returns a vector of all classified signs.
- static const std::vector< DetectedSignCombination > ∗const staticGetDetectedSignCombinations ()

  Returns a vector of all DetectedSignCombination}s.

Private Attributes

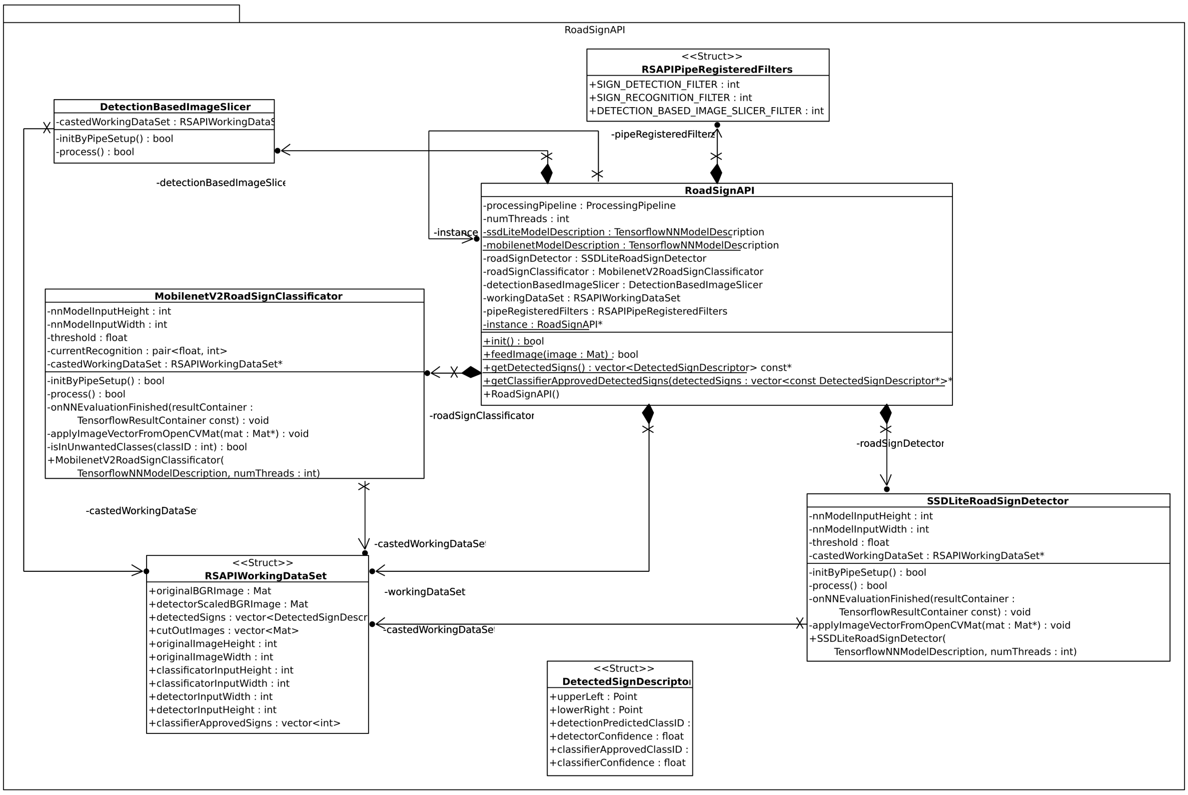
- FilterManagementLibrary::PipeSystem::ProcessingPipeline processingPipeline
- SSDLiteRoadSignDetector roadSignDetector
- MobilenetV2RoadSignClassificator roadSignClassificator
- DetectionBasedImageSlicer detectionBasedImageSlicer
- ClassifiedSignsGrouper classifiedSignsGrouper
- RoadSignDuplicationDeleter roadSignDuplicationDeleter
- RSAPIWorkingDataSet workingDataSet
- RSAPIPipeRegisteredFilters pipeRegisteredFilters
- AAssetManager ∗const assetManager

Static Private Attributes

- static FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription ssdLiteModelDescription
- static FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription mobilenetModelDescription
- static RoadSignAPI ∗ instance

7.13.1 Detailed Description

Implemenation of the RoadSignAPI. Here all filters used for roadsign detection and classification will be set up and managed. It provides a static interface using a static instance of RoadSignAPI to easily feed an image and analyze it from static environments (i.e. Java Native Interface)


RoadSignAPI
<<Struct>>
RSAPIPipeRegisteredFilters
+SIGN_DETECTION_FILTER : int
+SIGN_RECOGNITION_FILTER : int
+DETECTION_BASED_IMAGE_SLICER_FILTER : int
-pipeRegisteredFilters
DetectionBasedImageSlicer
-castedWorkingDataSet : RSAPIWorkingDataS
-initByPipeSetup() : bool
-process() : bool
-detectionBasedImageSlice
-instance
RoadSignAPI
-processingPipeline : ProcessingPipeline
-numThreads : int
-ssdLiteModelDescription : TensorflowNNModelDescription
-mobilenetModelDescription : TensorflowNNModelDescription
-roadSignDetector : SSDLiteRoadSignDetector
-roadSignClassificator : MobilenetV2RoadSignClassificator
-detectionBasedImageSlicer : DetectionBasedImageSlicer
-workingDataSet : RSAPIWorkingDataSet
-pipeRegisteredFilters : RSAPIPipeRegisteredFilters
-instance : RoadSignAPI*
+init() : bool
+feedImage(image : Mat) : bool
+getDetectedSigns() : vector<DetectedSignDescriptor> const*
+getClassifierApprovedDetectedSigns(detectedSigns : vector<const DetectedSignDescriptor*>*
+RoadSignAPI()
MobilenetV2RoadSignClassificator
-nnModelInputHeight : int
-nnModelInputWidth : int
-threshold : float
-currentRecognition : pair<float, int>
-castedWorkingDataSet : RSAPIWorkingDataSet*
-initByPipeSetup() : bool
-process() : bool
-onNNEvaluationFinished(resultContainer :
TensorflowResultContainer const) : void
-applyImageVectorFromOpenCVMat(mat : Mat*) : void
-isInUnwantedClasses(classID : int) : bool
+MobilenetV2RoadSignClassificator(
TensorflowNNModelDescription, numThreads : int)
-roadSignClassificator
-roadSignDetector
-castedWorkingDataSe
-castedWorkingDataSe
SSDLiteRoadSignDetector
-nnModelInputHeight : int
-nnModelInputWidth : int
-threshold : float
-castedWorkingDataSet : RSAPIWorkingDataSet*
-initByPipeSetup() : bool
-process() : bool
-onNNEvaluationFinished(resultContainer :
TensorflowResultContainer const) : void
-applyImageVectorFromOpenCVMat(mat : Mat*) : void
+SSDLiteRoadSignDetector(
TensorflowNNModelDescription, numThreads : int)
<<Struct>>
RSAPIWorkingDataSet
+originalBGRImage : Mat
+detectorScaledBGRImage : Mat
+detectedSigns : vector<DetectedSignDescr
+cutOutImages : vector<Mat>
+originalImageHeight : int
+originalImageWidth : int
+classificatorInputHeight : int
+classificatorInputWidth : int
+detectorInputWidth : int
+detectorInputHeight : int
+classifierApprovedSigns : vector<int>
-workingDataSet
-castedWorkingDataSe
<<Struct>>
DetectedSignDescripto
+upperLeft : Point
+lowerRight : Point
+detectionPredictedClassID :
+detectorConfidence : float
+classifierApprovedClassID :
+classifierConfidence : float

7.13.2 Constructor & Destructor Documentation

7.13.2.1 RoadSignAPI() [1/2]

```
RoadSignAPI::RoadSignAPI::RoadSignAPI (
            FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription detector←
ModelDescription,
            FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription classificator←
ModelDescription,
            const int numThreads,
            AAssetManager *const assetManager )
```

Constructor of the RoadSignAPI for Android environments.

Under Android, we need a slightly different constructor for the RoadSignAPI: As we want to store the neuronal network model files in the assets folder of the app, we need a Pointer to an AssetManager passed from the Java part of the app (by any means). The Android-platform dependent code of the RoadSignAPI expects this (also refer to SSDLiteRoadSignDetector and MobilenetV2RoadSignClassificator for more details). Of course this constructor is only available under Android environments. For other platforms, please use the corresponding constructor. Calls the constructor of the ProcessingPipeline and all filters. Afterwards the filters are registered to the pipe.

Parameters

TensorflowNNModelDescription	The model description for the detector network.
TensorflowNNModelDescription	The model description for the classificator network.
const	int numThreads Number of threads Tensorflow shall be allowed to use at max.
AAssetManager∗	const assetManager pointer to an AssetManager, which we will use to load neuronal network model files from the assets folder of the Android app. Needs to be passed from the Java part of the app by any means.

7.13.2.2 RoadSignAPI() [2/2]

```
RoadSignAPI::RoadSignAPI::RoadSignAPI (
            FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription detector←↩
ModelDescription,
            FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription classificator←↩
ModelDescription,
            const int numThreads )
```

Constructor of the RoadSignAPI for non Android environments.

Calls the constructor of the ProcessingPipeline and all filters. Afterwards the filters are registered to the pipe.

Parameters

TensorflowNNModelDescription	The model description for the detector network.
TensorflowNNModelDescription	The model description for the classificator network.
const	int numThreads Number of threads Tensorflow shall be allowed to use at max.

7.13.3 Member Function Documentation

7.13.3.1 feedImage()

```
bool RoadSignAPI::RoadSignAPI::feedImage (
            cv::Mat iamge )
```

Takes an OpenCV Mat uses the filter to examine it for road signs.

References image.

7.13.3.2 getClassifierApprovedDetectedSigns()

```
void RoadSignAPI::RoadSignAPI::getClassifierApprovedDetectedSigns (
             std::vector< const DetectedSignDescriptor *> * detectedSigns )
```

Returns a vector of all classified signs.

Should only be called after the feedImage(...) function was used. The vector contains all signs which were detected AND classified on the previous image. All images which were detected but could not be classified or are classified as a part of the unwanted classes won't be contained in this list!

Parameters

std::vector<const	DetectedSignDescriptor*>* a pointer to a vector of pointers to a DetectedSignDescriptor. The vector will be filled with the DetectedSignDescriptors of the RSAPIWorkingDataSet which were successfully classified as relevant roadsigns.

Returns

void

References i.

7.13.3.3 getDetectedSignCombinations()

```
const std::vector< RoadSignAPI::DetectedSignCombination > *const RoadSignAPI::RoadSignAPI↩
::getDetectedSignCombinations ( ) const
```

Returns a vector of all DetectedSignCombination}s.

Should only be called after the feedImage(...) function was used. The vector contains all signs which were detected AND classified on the previous image grouped into DetectedSignCombination}s. All images which were detected but could not be classified or are classified as a part of the unwanted classes won't be contained in this list!

Returns

const std::vector<DetectedSignCombination>* const pointer to a vector containing all DetectedSignCombination}s, that were found in the image that was lastly provided to feedImage(...).

7.13.3.4 getDetectedSigns()

```
const std::vector< RoadSignAPI::DetectedSignDescriptor > * RoadSignAPI::RoadSignAPI::get←↩
DetectedSigns ( )
```

Returns a vector of all detected (not classified!) signs.

Should only be called after the feedImage(...) function was used. The vector contains all signs which were detected on the previous image.

Returns

const std::vector<DetectedSignDescriptor> a vector containing all detected signs.

7.13.3.5 init()

```
bool RoadSignAPI::RoadSignAPI::init ( )
```

Initializes the RoadSignAPI's static interface.

We use a Singleton to provide a static interface to the RoadSignAPI. This Singleton will be initialized in this function: The model descriptions for the SSDLite and MobilenetV2 neuronal networks will be created, the RoadSignAPI will be instantiated and the ProcessingPipeline's setUp function is called.

7.13.3.6 staticFeedImage()

```
bool RoadSignAPI::RoadSignAPI::staticFeedImage (
            cv::Mat image )  [static]
```

Takes an OpenCV Mat uses the filter to examine it for road signs.

References feedImage().

7.13.3.7 staticGetClassifierApprovedDetectedSigns()

```
void RoadSignAPI::RoadSignAPI::staticGetClassifierApprovedDetectedSigns (
            std::vector< const DetectedSignDescriptor *> * detectedSigns )  [static]
```

Returns a vector of all classified signs.

Should only be called after the staticFeedImage(...) function was used. The vector contains all signs which were detected AND classified on the previous image. All images which were detected but could not be classified or are classified as a part of the unwanted classes won't be contained in this list!

Parameters

std::vector<const	DetectedSignDescriptor∗>∗ a pointer to a vector of pointers to a DetectedSignDescriptor. The vector will be filled with the DetectedSignDescriptors of the RSAPIWorkingDataSet which were successfully classified as relevant roadsigns.

Returns

void

7.13.3.8 staticGetDetectedSignCombinations()

```
const std::vector< RoadSignAPI::DetectedSignCombination > *const RoadSignAPI::RoadSignAPI←
::staticGetDetectedSignCombinations ( )  [static]
```

Returns a vector of all DetectedSignCombination}s.

Should only be called after the staticFeedImage(...) function was used. The vector contains all signs which were detected AND classified on the previous image grouped into DetectedSignCombination}s. All images which were detected but could not be classified or are classified as a part of the unwanted classes won't be contained in this list!

Returns

const std::vector<DetectedSignCombination>∗ const pointer to a vector containing all DetectedSign←Combination}s, that were found in the image that was lastly provided to staticFeedImage(...).

References getDetectedSignCombinations().

7.13.3.9 staticGetDetectedSigns()

```
const std::vector< RoadSignAPI::DetectedSignDescriptor > * RoadSignAPI::RoadSignAPI::static←
GetDetectedSigns ( )  [static]
```

Returns a vector of all detected (not classified!) signs.

Should only be called after the feedImage(...) function was used. The vector contains all signs which were detected on the previous image.

Returns

const std::vector<DetectedSignDescriptor> a vector containing all detected signs.

References getDetectedSigns().

7.13.3.10 staticInit() [1/2]

```
bool RoadSignAPI::RoadSignAPI::staticInit (
            const int numThreads,
            AAssetManager *const assetManager )  [static]
```

Initializes the RoadSignAPI's static interface.

This is the staticInit function for Android environments. As under Android, we want the neuronal network model files to be stored inside the assets folder of the app, we need an AssetManager passed from the Java part of the app to be able to do so. On non Android environments, this function just lacks it's last parameter.

We use a Singleton to provide a static interface to the RoadSignAPI. This Singleton will be initialized in this function: The model descriptions for the SSDLite and MobilenetV2 neuronal networks will be created, the RoadSignAPI will be instantiated and the ProcessingPipeline's setUp function is called.

References init().

7.13.3.11 staticInit() [2/2]

```
bool RoadSignAPI::RoadSignAPI::staticInit (
            const int numThreads )  [static]
```

Initializes the RoadSignAPI's static interface.

This is the staticInit function for non-Android environments. We use a Singleton to provide a static interface to the RoadSignAPI. This Singleton will be initialized in this function: The model descriptions for the SSDLite and MobilenetV2 neuronal networks will be created, the RoadSignAPI will be instantiated and the ProcessingPipeline's setUp function is called.

References init().

7.13.4 Member Data Documentation

7.13.4.1 assetManager

```
AAssetManager* const RoadSignAPI::RoadSignAPI::assetManager  [private]
```

A pointer to an AssetManager which can be passed via the constructor. We need it to be able to load model files from the Android assets folder, which we want to do so the files do not have to be placed in the normal user space. Will be passed to SSDLiteRoadSignDetector and MobilenetV2RoadSignClassificator.

7.13.4.2 classifiedSignsGrouper

```
ClassifiedSignsGrouper RoadSignAPI::RoadSignAPI::classifiedSignsGrouper  [private]
```

Instance of ClassifiedSignsGrouper which groups all signs that where detected *AND* classified into DetectedSignCombination}s

7.13.4.3 detectionBasedImageSlicer

```
DetectionBasedImageSlicer RoadSignAPI::RoadSignAPI::detectionBasedImageSlicer [private]
```

Instance of DetectionBasedImageSlicer used to crop the boxes in which the roadsigns are contained, predicted by the SSDLiteRoadSignDetector.

7.13.4.4 instance

```
RoadSignAPI::RoadSignAPI * RoadSignAPI::RoadSignAPI::instance [static], [private]
```

Static instance of RoadSignAPI for static interface.

7.13.4.5 mobilenetModelDescription

```
FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription RoadSignAPI::RoadSignAPI::mobilenetModelDescription [static], [private]
```

Static description for the mobilenetv2 network, used for static interface.

7.13.4.6 pipeRegisteredFilters

```
RSAPIPipeRegisteredFilters RoadSignAPI::RoadSignAPI::pipeRegisteredFilters [private]
```

Instance of RSAPIPipeRegisteredFilters, which is a struct in which all IDs of the filters which are added to the ProcessingPipeline of this class are stored. It is passed to all filters so they know the IDs of all other filters.

7.13.4.7 processingPipeline

```
FilterManagementLibrary:: PipeSystem::ProcessingPipeline RoadSignAPI::RoadSignAPI::processingPipeline [private]
```

Pipeline which is used for sequencial processing of filters used for roadsign detection, classification, grouping and filtering.

7.13.4.8 roadSignClassificator

```
MobilenetV2RoadSignClassificator RoadSignAPI::RoadSignAPI::roadSignClassificator [private]
```

Instance of the MobilenetV2RoadSignClassificator used for roadsign classification, which is registered to the pipe of an instance of RoadSignAPI.

7.13.4.9 roadSignDetector

```
SSDLiteRoadSignDetector RoadSignAPI::RoadSignAPI::roadSignDetector [private]
```

Instance of the SSDLiteRoadSignDetector used for roadsign detection, which is registered to the pipe of an instance of RoadSignAPI.

7.13.4.10 roadSignDuplicationDeleter

```
RoadSignDuplicationDeleter RoadSignAPI::RoadSignAPI::roadSignDuplicationDeleter  [private]
```

Instance of RoadSignDuplicationDeleter which is used to delete signs that were detected multiple times (with a little offset) by the SSDLiteRoadSignDetector.

7.13.4.11 ssdLiteModelDescription

```
FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription RoadSignAPI::RoadSignAP←
I::ssdLiteModelDescription  [static], [private]
```

Static description for the ssdLite network, used for static interface.

7.13.4.12 workingDataSet

```
RSAPIWorkingDataSet RoadSignAPI::RoadSignAPI::workingDataSet  [private]
```

Instance of RSAPIWorkingDataSet, which is the shared resource in which all results of the filters used in this class are stored.

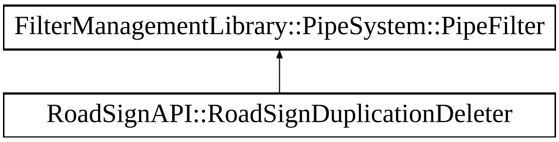
The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/RoadSignAPI.h
- RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/RoadSignAPI.cpp

7.14 RoadSignAPI::RoadSignDuplicationDeleter Class Reference

```
#include <RoadSignDuplicationDeleter.h>
```

Inheritance diagram for RoadSignAPI::RoadSignDuplicationDeleter:


FilterManagementLibrary::PipeSystem::PipeFilter
RoadSignAPI::RoadSignDuplicationDeleter

Private Member Functions

- void markForDeletion (int ID)
- void deleteAllMarkedSignDescriptors ()
- bool isMarkedForDeletion (int ID) const
- bool signsDoOverlap (DetectedSignDescriptor ∗first, DetectedSignDescriptor ∗second) const
- float calculateOverlapPercentage (DetectedSignDescriptor ∗first, DetectedSignDescriptor ∗second) const
- int calculateBoxArea (DetectedSignDescriptor ∗signDescriptor) const
- bool initByPipeSetup ()

  Initializes the filter.
- bool process ()

  The process function of this filter.

Private Attributes

- RSAPIWorkingDataSet ∗ castedWorkingDataSet
- float minOverlapPercentage = 0.60
- std::vector< int > signsToDelete

Additional Inherited Members

7.14.1 Detailed Description

As it is possible that the detector detects one RoadSign at about the same position (with a small offset), it is neccessary to filter those duplicates out, because they would be classified and added to a DetectedSignCombination (by ClassifiedSignsGrouper) twice. This Filter tries to the detect duplicates by checking the congruency of the rectangles of the detected signs. If they overlap by a certain percentage (specified by minOverLapPercentage), the bigger sign will be dropped. Why the bigger sign? Because it is possible that the detector detects two signs three times: two times as single signs, and the third time combined into one sign, which is - obviously - not wanted.

7.14.2 Member Function Documentation

7.14.2.1 calculateBoxArea()

```
int RoadSignAPI::RoadSignDuplicationDeleter::calculateBoxArea (
            DetectedSignDescriptor ∗ signDescriptor ) const  [private]
```

Calculates the area of the box of a DetectedSignDescriptor}

Returns

int width ∗ height as the area of the box of the signDescriptor.

References RoadSignAPI::DetectedSignDescriptor::lowerRight, and RoadSignAPI::DetectedSignDescriptor::upperLeft.

7.14.2.2 calculateOverlapPercentage()

```
float RoadSignAPI::RoadSignDuplicationDeleter::calculateOverlapPercentage (
            DetectedSignDescriptor ∗ first,
            DetectedSignDescriptor ∗ second ) const  [private]
```

Calculates the percentage of which the box (rectangle) of the first DetectedSignDescriptor} does overlap with the second (the order of first and second is relevant ! The percentage is calculated relative of the area of the box of first).

Returns

float Percentage of the overlapping area of the boxes of the two signs relative to the area of the box of first.

References calculateBoxArea(), RoadSignAPI::DetectedSignDescriptor::lowerRight, and RoadSignAPI::DetectedSignDescriptor::upperLeft.

7.14.2.3 deleteAllMarkedSignDescriptors()

```
void RoadSignAPI::RoadSignDuplicationDeleter::deleteAllMarkedSignDescriptors ( )  [private]
```

Deletes all signs in castedWorkingDataSet->detectedSigns, which were marked for deletion during the process() function. This may change the order of the detectedSigns list, because we want to avoid bigger copy and / or move operations. So the element which shall be deleted is overriden by the last element of the list, and afterwards the list is reduced by one.

References castedWorkingDataSet, RoadSignAPI::RSAPIWorkingDataSet::detectedSigns, i, and signsToDelete.

7.14.2.4 initByPipeSetup()

```
bool RoadSignAPI::RoadSignDuplicationDeleter::initByPipeSetup ( )  [private], [virtual]
```

Initializes the filter.

As this filter does not need any specific environment variables, nothing really exciting is happenning here..

Returns

true, always

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References castedWorkingDataSet, FilterManagementLibrary::PipeSystem::PipeFilter::pipeWorkingDataSet, and FilterManagementLibrary::Logger::printfln().

7.14.2.5 isMarkedForDeletion()

```
bool RoadSignAPI::RoadSignDuplicationDeleter::isMarkedForDeletion (
             int ID ) const  [private]
```

References signsToDelete.

7.14.2.6 markForDeletion()

```
void RoadSignAPI::RoadSignDuplicationDeleter::markForDeletion (
             int ID )  [private]
```

Marks the ID of the sign for deletion, thus adding it to the signsToDelete (member variable) list.

References signsToDelete.

7.14.2.7 process()

```
bool RoadSignAPI::RoadSignDuplicationDeleter::process ( )  [private], [virtual]
```

The process function of this filter.

Filters duplicates of signs (that were detected multiple times with an offset). For this it iterates over the list of all signs and checks each sign against each other sign and calculates the percentage of the overlap of both rectangles (of the signs). If they overlap by at least minOverlapPercentage (member variable), than the bigger sign is dropped. Why the bigger sign, you may ask? Because it is likely that, if we have one big detected box and one smaller box in it, there are actually two signs inside the box. If that's the case it is likely that also the second smaller sign is detected another time (so we have three detections: two smaller ones, each representing one sign, and one bigger which contains the two signs combined), so those two smaller detections shall be keeped. And if the second smaller sign is not detected, the bigger box would probably be false anyways.

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References calculateBoxArea(), calculateOverlapPercentage(), castedWorkingDataSet, deleteAllMarkedSign↩Descriptors(), RoadSignAPI::RSAPIWorkingDataSet::detectedSigns, i, FilterManagementLibrary::PipeSystem::↩PipeFilter::invokeNext(), isMarkedForDeletion(), markForDeletion(), minOverlapPercentage, FilterManagement↩Library::PipeSystem::PipeFilter::pipeRegisteredFilters, signsDoOverlap(), and signsToDelete.

7.14.2.8 signsDoOverlap()

```
bool RoadSignAPI::RoadSignDuplicationDeleter::signsDoOverlap (
            DetectedSignDescriptor * first,
            DetectedSignDescriptor * second ) const  [private]
```

Checks if the boxes (rectangles) of two DetectedSignDescriptor}s do overlap by doing some basic mathematical operations using the position and dimension of the boxes.

Returns

true, if they do overlap at least by one pixel, false otherwise

References RoadSignAPI::DetectedSignDescriptor::lowerRight, and RoadSignAPI::DetectedSignDescriptor↩::upperLeft.

7.14.3 Member Data Documentation

7.14.3.1 castedWorkingDataSet

```
RSAPIWorkingDataSet* RoadSignAPI::RoadSignDuplicationDeleter::castedWorkingDataSet  [private]
```

Just a pointer casted from PipeWorkingDataSet∗ to RSAPIWorkingDataSet∗, so we just don't have to do the casting every time we need it ;)

7.14.3.2 minOverlapPercentage

```
float RoadSignAPI::RoadSignDuplicationDeleter::minOverlapPercentage = 0.60  [private]
```

The minimum percentage by which two signs need to overlap to be considered equal.

7.14.3.3 signsToDelete

```
std::vector<int> RoadSignAPI::RoadSignDuplicationDeleter::signsToDelete  [private]
```

A local list which is used by markForDeletion(...), deleteAllMarkedSignDescriptors() and isMarkedForDeletion(...). It is used to remember which of the signs which were evaluated being equal in the process() function shall be deleted, because they are duplicates.

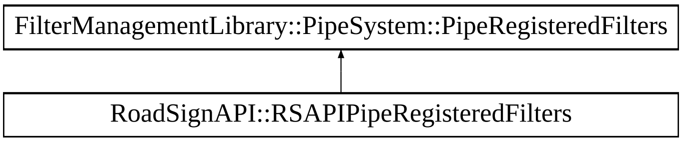
The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/RoadSignDuplicationDeleter.h
- RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/RoadSignDuplicationDeleter.cpp

7.15 RoadSignAPI::RSAPIPipeRegisteredFilters Struct Reference

```
#include <RSAPIPipeRegisteredFilters.h>
```

Inheritance diagram for RoadSignAPI::RSAPIPipeRegisteredFilters:


FilterManagementLibrary::PipeSystem::PipeRegisteredFilters
RoadSignAPI::RSAPIPipeRegisteredFilters

Public Attributes

- int SIGN_DETECTION_FILTER
- int SIGN_RECOGNITION_FILTER
- int SIGN_DUPLICATION_DELETER_FILTER
- int DETECTION_BASED_IMAGE_SLICER_FILTER
- int CLASSIFIED_SIGNS_GROUPER_FILTER

7.15.1 Detailed Description

Contains variables which will hold the ID's of the filters of the API. See PipeWorkingDataSet for a detailed description of the the intention of this struct.

7.15.2 Member Data Documentation

7.15.2.1 CLASSIFIED_SIGNS_GROUPER_FILTER

```
int RoadSignAPI::RSAPIPipeRegisteredFilters::CLASSIFIED_SIGNS_GROUPER_FILTER
```

7.15.2.2 DETECTION_BASED_IMAGE_SLICER_FILTER

```
int RoadSignAPI::RSAPIPipeRegisteredFilters::DETECTION_BASED_IMAGE_SLICER_FILTER
```

7.15.2.3 SIGN_DETECTION_FILTER

```
int RoadSignAPI::RSAPIPipeRegisteredFilters::SIGN_DETECTION_FILTER
```

7.15.2.4 SIGN_DUPLICATION_DELETER_FILTER

```
int RoadSignAPI::RSAPIPipeRegisteredFilters::SIGN_DUPLICATION_DELETER_FILTER
```

7.15.2.5 SIGN_RECOGNITION_FILTER

```
int RoadSignAPI::RSAPIPipeRegisteredFilters::SIGN_RECOGNITION_FILTER
```

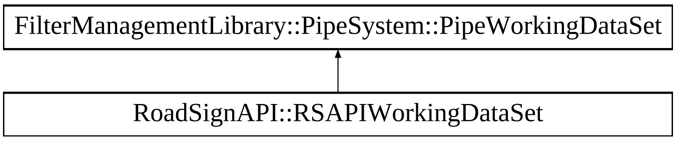
The documentation for this struct was generated from the following file:

- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/RSAPIPipeRegisteredFilters/RSAPIPipeRegisteredFilters.h

7.16 RoadSignAPI::RSAPIWorkingDataSet Struct Reference

```
#include <RSAPIWorkingDataSet.h>
```

Inheritance diagram for RoadSignAPI::RSAPIWorkingDataSet:


FilterManagementLibrary::PipeSystem::PipeWorkingDataSet
RoadSignAPI::RSAPIWorkingDataSet

Public Attributes

- cv::Mat originalBGRImage
- cv::Mat detectorScaledBGRImage
- std::vector< DetectedSignDescriptor > detectedSigns
- std::vector< cv::Mat > cutOutImages
- int originalImageHeight
- int originalImageWidth
- std::vector< int > classifierApprovedSigns
- std::vector< DetectedSignCombination > detectedSignCombinations

7.16.1 Detailed Description

Contains all variables which are needed to (efficiently!) detect and classifiy roadsigns from an OpenCV image Matrix. The detected and classified signs will also be stored here as a result.

7.16.2 Member Data Documentation

7.16.2.1 classifierApprovedSigns

```
std::vector<int> RoadSignAPI::RSAPIWorkingDataSet::classifierApprovedSigns
```

Contains the ID of entries in detectedSigns which were successfully classified by the classifier Filter and are not in the class of unimportant signs.

7.16.2.2 cutOutImages

```
std::vector<cv::Mat> RoadSignAPI::RSAPIWorkingDataSet::cutOutImages
```

This vector will be filled by the DetectionBasedImageSlicer. It uses the detectedSigns vector to crop the corresponding signs from the originalBgrImage. MobilenetV2RoadSignClassificator will scale them to the size it needs using bicubic interpolation.

7.16.2.3 detectedSignCombinations

```
std::vector<DetectedSignCombination> RoadSignAPI::RSAPIWorkingDataSet::detectedSignCombinations
```

The ClassifiedSignsGrouper tries to group all successfully classified signs on an (estimated) pole (which means they are located vertically above each other) into an DetectedSignCombination. The result is stored in this vector.

7.16.2.4 detectedSigns

```
std::vector<DetectedSignDescriptor> RoadSignAPI::RSAPIWorkingDataSet::detectedSigns
```

Describes the signs that were detected by the SSDLiteRoadSignDetector. See DetectedSignDescriptor for a more detailed description.

7.16.2.5 detectorScaledBGRImage

```
cv::Mat RoadSignAPI::RSAPIWorkingDataSet::detectorScaledBGRImage
```

This will be initialized by SSDLiteRoadSignDetector. It creates a scaled cv::Mat from the originalBGRImage (in 8UC3 format, too) which matches the input size of the neuronal network model used for the SSDLiteRoadSignDetector using bicubic interpolation.

7.16.2.6 originalBGRImage

```
cv::Mat RoadSignAPI::RSAPIWorkingDataSet::originalBGRImage
```

This is the image which was feed into the API using {see RoadSignAPI}'s feedImage(...) function Needs to be 8UC3 BGR Image because {link TensorflowOpenCVUtils} expects that.

7.16.2.7 originalImageHeight

```
int RoadSignAPI::RSAPIWorkingDataSet::originalImageHeight
```

Height of the originalBGRImage.

7.16.2.8 originalImageWidth

```
int RoadSignAPI::RSAPIWorkingDataSet::originalImageWidth
```

Width of the originalBGRImage

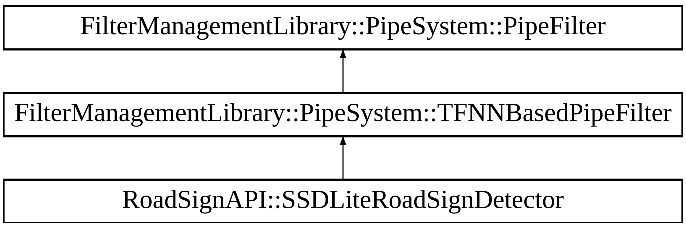
The documentation for this struct was generated from the following file:

- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/RSAPIWorkingDataSet/RSAPIWorkingDataSet.h

7.17 RoadSignAPI::SSDLiteRoadSignDetector Class Reference

```
#include <SSDLiteRoadSignDetector.h>
```

Inheritance diagram for RoadSignAPI::SSDLiteRoadSignDetector:


FilterManagementLibrary::PipeSystem::PipeFilter
FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter
RoadSignAPI::SSDLiteRoadSignDetector

Public Member Functions

- SSDLiteRoadSignDetector (FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription, int numThreads, AAssetManager ∗const assetManager)

  Constructor of SSDLiteRoadSignDetector for Android environments. It basically just calls the TFNNBasedPipeFilter super constructor. Refer to it for a more detailed description. Under Android, the RoadSignAPI expects the model files of our neuronal networks to be placed in the assets folder of the app. Thus, this constructor needs to be passed a pointer to an AssetManager (from java), which is MANDATORY for the RoadSignAPI under Android!
- SSDLiteRoadSignDetector (FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription, int numThreads)

  Constructor of SSDLiteRoadSignDetector It basically just calls the TFNNBasedPipeFilter super constructor. Refer to it for a more detailed description.

Private Member Functions

- bool initByPipeSetup ()

  Initializes the filter.
- bool process ()

  Uses the neuronal network to detect signs on the current image.
- void onNNEvaluationFinished (const FilterManagementLibrary::TFIntegration::TensorflowResultContainer resultContainer)

  Callback, will be called when the network finished it's prediction.
- void applyImageVectorFromOpenCVMat (cv::Mat ∗mat)

  Uses TensorflowOpenCVUtils to apply a mat as input to the network.

Private Attributes

- int nnModelInputHeight
- int nnModelInputWidth
- float threshold = 0.28f
- RSAPIWorkingDataSet ∗ castedWorkingDataSet
- AAssetManager ∗const assetManager

Additional Inherited Members

7.17.1 Detailed Description

This filter is based on TFNNBasedPipeFilter and uses SSDLite for (roadsign) detection.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 SSDLiteRoadSignDetector() [1/2]

```
RoadSignAPI::SSDLiteRoadSignDetector::SSDLiteRoadSignDetector (
              FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription nnModel←
Description,
              int numThreads,
              AAssetManager ∗const assetManager )
```

Constructor of SSDLiteRoadSignDetector for Android environments. It basically just calls the TFNNBasedPipeFilter super constructor. Refer to it for a more detailed description. Under Android, the RoadSignAPI expects the model files of our neuronal networks to be placed in the assets folder of the app. Thus, this constructor needs to be passed a pointer to an AssetManager (from java), which is MANDATORY for the RoadSignAPI under Android!

All other class-member variables will be initialized to their default values.

Parameters

TensorflowNNModelDescription	nnModelDescription a description of the Tensorflow model which will be used by the filter.
int	numThreads number of threads Tensorflow is allowed to use for computation.
AAssetManager∗	const assetManager pointer to an Android Asset Manager which needs to be passed from the Java part of the Android App by any means. Manadatory to load the neuronal network models from the assets directory of the app.

7.17.2.2 SSDLiteRoadSignDetector() [2/2]

```
RoadSignAPI::SSDLiteRoadSignDetector::SSDLiteRoadSignDetector (
             FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription nnModel←↩
Description,
             int numThreads )
```

Constructor of SSDLiteRoadSignDetector It basically just calls the TFNNBasedPipeFilter super constructor. Refer to it for a more detailed description.

All class-member variables will be initialized to their default values.

Parameters

TensorflowNNModelDescription	nnModelDescription a description of the Tensorflow model which will be used by the filter.
int	numThreads number of threads Tensorflow is allowed to use for computation.

7.17.3 Member Function Documentation

7.17.3.1 applyImageVectorFromOpenCVMat()

```
void RoadSignAPI::SSDLiteRoadSignDetector::applyImageVectorFromOpenCVMat (
             cv::Mat ∗ mat ) [private]
```

Uses TensorflowOpenCVUtils to apply a mat as input to the network.

Parameters

cv::Mat	∗mat pointer to an OpenCV Mat which shall be used as input.

Returns

void

References FilterManagementLibrary::TensorflowOpenCVUtils::fastApplyCVMatOnInputTensorUInt8(), and FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::getNNInputTensor().

7.17.3.2 initByPipeSetup()

```
bool RoadSignAPI::SSDLiteRoadSignDetector::initByPipeSetup ( )  [private], [virtual]
```

Initializes the filter.

Will be called by ProcessingPipeline. Here, basically just the neuronal network model will be loaded. For this, the setupModel() function will be called, refer to TFNNBasedPipeFilter for a more detailed description. Also the RSAPIWorkingDataSet's detectorSgaledBGRImage will be initialized (with the corresponding size and type).

Returns

true, if the model could be loaded successfully, flase otherwise

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References assetManager, castedWorkingDataSet, RoadSignAPI::RSAPIWorkingDataSet::detectorScaledBGRImage, FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::getNNModelDescription(), FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputHeight, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputWidth, nnModelInputHeight, nnModelInputWidth, FilterManagementLibrary::PipeSystem::PipeFilter::pipeWorkingDataSet, FilterManagementLibrary::Logger::printfln(), FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::setupModelFromAssets(), and FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::setupModelFromFile().

7.17.3.3 onNNEvaluationFinished()

```
void RoadSignAPI::SSDLiteRoadSignDetector::onNNEvaluationFinished (
             const FilterManagementLibrary::TFIntegration::TensorflowResultContainer resultContainer )  [private], [virtual]
```

Callback, will be called when the network finished it's prediction.

In process(), the TFNNBasedPipeFilter's (super class) evaluteInputVectorByNN() will be called. If it was successfull, this callback will be called and passed a TensorflowResultContainer with the result of the inference of the neuronal network model. Here, we will interpete the output of the network. We iterate over all it's detections and add them to a DetectedSignDescriptor, if the cofidence exceeds a certain threshold value. This descriptor is then added to RSAPIWorkingDataSet.

Parameters

const	TensorflowResultContainer resultContainer contains the results (in Tensorflow tensors) of the network prediction.

Implements FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter.

References applyImageVectorFromOpenCVMat(), castedWorkingDataSet, RoadSignAPI::RSAPIWorkingDataSet::detectedSigns, RoadSignAPI::DetectedSignDescriptor::detectionPredictedClassID, RoadSignAPI::DetectedSignDescriptor::detectorConfidence, i, FilterManagementLibrary::PipeSystem::PipeFilter::indicateProcessingFinished(), RoadSignAPI::DetectedSignDescriptor::lowerRight, threshold, and RoadSignAPI::DetectedSignDescriptor::upperLeft.

7.17.3.4 process()

```
bool RoadSignAPI::SSDLiteRoadSignDetector::process ( )  [private], [virtual]
```

Uses the neuronal network to detect signs on the current image.

The originalBGR image will be scaled to the size the network model expects. Afterfards, it uses the TFNNBasedPipeFilter super class's evaluateInputVectorByNN() function to detect the road signs.

Returns

> true if evaluateInputVectorByNN() return true, false otherwise (does NOT return false if no signs were detected!)

Implements FilterManagementLibrary::PipeSystem::PipeFilter.

References applyImageVectorFromOpenCVMat(), castedWorkingDataSet, RoadSignAPI::RSAPIWorkingDataSet::detectedSignCombinations, RoadSignAPI::RSAPIWorkingDataSet::detectedSigns, RoadSignAPI::RSAPIWorkingDataSet::detectorScaledBGRImage, FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::evaluateInputVectorByNN(), FilterManagementLibrary::PipeSystem::PipeFilter::invokeNext(), onNNEvaluationFinished(), RoadSignAPI::RSAPIWorkingDataSet::originalBGRImage, FilterManagementLibrary::PipeSystem::PipeFilter::pipeRegisteredFilters, and FilterManagementLibrary::Logger::printfln().

7.17.4 Member Data Documentation

7.17.4.1 assetManager

```
AAssetManager* const RoadSignAPI::SSDLiteRoadSignDetector::assetManager  [private]
```

A pointer to an AssetManager which can be passed via the constructor (only available under Android environments). We need it to be able to load model files from the Android assets folder, which we want to do so the files do not have to be placed in the normal user space.

7.17.4.2 castedWorkingDataSet

```
RSAPIWorkingDataSet* RoadSignAPI::SSDLiteRoadSignDetector::castedWorkingDataSet  [private]
```

Just a pointer casted from PipeWorkingDataSet* to RSAPIWorkingDataSet*, so we just don't have to do the casting every time we need it ;)

7.17.4.3 nnModelInputHeight

```
int RoadSignAPI::SSDLiteRoadSignDetector::nnModelInputHeight  [private]
```

Just a copy from the input height provided in the TensorflowNNModelDescription of the underlaying TensorflowN←NInstance.

7.17.4.4 nnModelInputWidth

```
int RoadSignAPI::SSDLiteRoadSignDetector::nnModelInputWidth  [private]
```

Just a copy from the input height provided in the TensorflowNNModelDescription of the underlaying TensorflowN←NInstance.

7.17.4.5 threshold

```
float RoadSignAPI::SSDLiteRoadSignDetector::threshold = 0.28f  [private]
```

Threshold the confidence of a prediction from our underlying neuronal network model needs to exceed in order to be included to RSAPIWorkingDataSet::detectedSigns

The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/SSDLiteRoadSignDetector.h
- RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/SSDLiteRoadSignDetector.cpp

7.18 FilterManagementLibrary::TensorflowAndroidJNIUtils Class Reference

```
#include <TensorflowAndroidJNIUtils.h>
```

Static Public Member Functions

- static bool PortableReadFileToProto (const std::string &file_name, ::google::protobuf::MessageLite ∗proto)
- static bool IsAsset (std::string filename)
- static void ReadFileToProto (AAssetManager ∗const asset_manager, std::string filename, google::protobuf←::MessageLite ∗message)
- static void ReadFileToString (AAssetManager ∗const asset_manager, std::string filename, std::string ∗str)
- static void ReadFileToVector (AAssetManager ∗const asset_manager, std::string filename, std::vector< std←::string > ∗str_vector)
- static void WriteProtoToFile (std::string filename, const google::protobuf::MessageLite &message)

7.18.1 Member Function Documentation

7.18.1.1 IsAsset()

```
bool FilterManagementLibrary::TensorflowAndroidJNIUtils::IsAsset (
            std::string filename )  [static]
```

References ASSET_PREFIX.

7.18.1.2 PortableReadFileToProto()

```
bool FilterManagementLibrary::TensorflowAndroidJNIUtils::PortableReadFileToProto (
            const std::string & file_name,
            ::google::protobuf::MessageLite * proto )  [static]
```

7.18.1.3 ReadFileToProto()

```
void FilterManagementLibrary::TensorflowAndroidJNIUtils::ReadFileToProto (
            AAssetManager *const asset_manager,
            std::string filename,
            google::protobuf::MessageLite * message )  [static]
```

References ASSET_PREFIX, IsAsset(), and FilterManagementLibrary::Logger::printfln().

7.18.1.4 ReadFileToString()

```
void FilterManagementLibrary::TensorflowAndroidJNIUtils::ReadFileToString (
            AAssetManager *const asset_manager,
            std::string filename,
            std::string * str )  [static]
```

References ASSET_PREFIX, IsAsset(), and FilterManagementLibrary::Logger::printfln().

7.18.1.5 ReadFileToVector()

```
void FilterManagementLibrary::TensorflowAndroidJNIUtils::ReadFileToVector (
            AAssetManager *const asset_manager,
            std::string filename,
            std::vector< std::string > * str_vector )  [static]
```

References FilterManagementLibrary::Logger::printfln().

7.18.1.6 WriteProtoToFile()

```
void FilterManagementLibrary::TensorflowAndroidJNIUtils::WriteProtoToFile (
            std::string filename,
            const google::protobuf::MessageLite & message )  [static]
```

References FilterManagementLibrary::Logger::printfln().

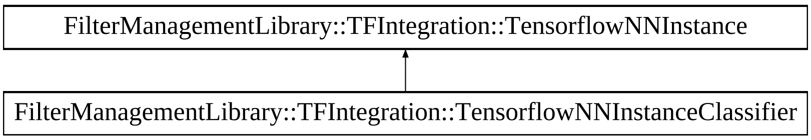
The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/TensorflowAndroidJNIUtils.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/TensorflowAndroidJNIUtils.cpp

7.19 FilterManagementLibrary::TFIntegration::TensorflowNNInstance Class Reference

```
#include <TensorflowNNInstance.h>
```

Inheritance diagram for FilterManagementLibrary::TFIntegration::TensorflowNNInstance:


FilterManagementLibrary::TFIntegration::TensorflowNNInstance
FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier

Public Types

- enum ErrorType {
  ErrorType::ERROR_NONE, ErrorType::ERROR_INVALID_MODEL_FILE, ErrorType::ERROR_FAILED_TO_CONSTRUCT_NEW_SESSION, ErrorType::ERROR_COULD_NOT_LOAD_MODEL,
  ErrorType::ERROR_COULD_NOT_ADD_GRAPH_TO_SESSION, ErrorType::ERROR_SESSION_RUN_FAILED, ErrorType::ERROR_INPUT_SIZE_MISMATCH, ErrorType::READ_FILE_TO_PROTO_FROM_ASSETS_FAILED,
  ErrorType::ERROR_TFNN_CLASSIFIER_INVALID_LABELS_FILE }

Public Member Functions

- bool setupModelFromFile ()

  Sets up the Tensorflow model using the provieded model description.
- bool setupModelFromAssets (AAssetManager *const assetManager)

  Sets up the Tensorflow model using the provieded model description.
- bool runInference ()

  Let's the neuronal network examine the applied input.
- bool applyImageInputVector (uint8_t ***inputVector, int imageHeight, int imageWidth, int channels)

  Applies an image vector to the instance of the Tensorflow model.
- bool applyInputVectorFromFloatData (float *floatData, const int heigth, const int width, const int channels)

  Applies an already float input vector to the neuronal network model.
- void adjustModelFile (std::string modelFile)

  Allows to change ONLY the model file path set in the model description.

- tensorflow::Tensor ∗ getInputTensor ()

  Returns the input Tensor of this instance.
- std::vector< tensorflow::Tensor > ∗ getOutputTensors ()

  Returns the tensors containg the output of the previous inference.
- TensorflowResultContainer getResultContainer ()

  Returns a container containing the result of the last inference.
- const TensorflowNNModelDescription ∗ getModelDescription () const

  Returns the model description which was passed to this instance.
- ErrorType getLastError () const

  Returns the last error that happened.
- TensorflowNNInstance (TensorflowNNModelDescription nnModelDescription, int numThreads=1)

  Constructor of TensorflowNNInstance.
- ∼TensorflowNNInstance ()

  Destructor of the TensorflowNNInstance class.

Protected Attributes

- tensorflow::Tensor inputTensor
- std::vector< tensorflow::Tensor > outputTensors
- std::unique_ptr< tensorflow::Session > tensorflowSession
- tensorflow::SessionOptions sessionOptions
- tensorflow::GraphDef graphDef
- std::unique_ptr< std::string > inputLayerName
- ErrorType lastError

Private Attributes

- TensorflowNNModelDescription nnModelDescription
- int numThreads = 1

7.19.1 Detailed Description

Provides all the functionality needed to load tensorflow models, add them to a tensorflow session, provide an input vector, functions to run inferences and methods to process the output.

7.19.2 Member Enumeration Documentation

7.19.2.1 ErrorType

```
enum FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ErrorType  [strong]
```

Enumerator

ERROR_NONE	Standard value set on initialization. Needed because getLastError() shall not return a random value (which could cause unexpected behavior)
ERROR_INVALID_MODEL_FILE	If the model file does not exist at the given path.
ERROR_FAILED_TO_CONSTRUCT_NEW_SESS↩ ION	Tensorflow related error. Thrown, if the NewSession() function in setupModel() fails.
ERROR_COULD_NOT_LOAD_MODEL	If the given model file could not be loaded because it is in wrong format or we do not have sufficient permission to read it.
ERROR_COULD_NOT_ADD_GRAPH_TO_SESSI↩ ON	Tensorflow related error. Thrown, if the function Create(...) of the Tensorflow Session fails with the graph definition build from the model file.
ERROR_SESSION_RUN_FAILED	Tensorflow related error. Thrown, if we could not run the session (maybe incorrect model file, missing ops in the Tensorflow library or any other configuration error).
ERROR_INPUT_SIZE_MISMATCH	If the dimensions of the given input vector do not match with the input sized in the provided TensorflowNNModelDescription.
READ_FILE_TO_PROTO_FROM_ASSETS_FAILED	Only available under Android environments. If using the setupModelFromAssets(...) function, but the modelFile provided in TensorflowNNModelDescription could not be loaded (for example is not found in the assets folder).
ERROR_TFNN_CLASSIFIER_INVALID_LABELS_↩ FILE	Only relevant for TensorflowNNInstanceClassifier if the specified labels file does not exist and thus could not be loaded.

7.19.3 Constructor & Destructor Documentation

7.19.3.1 TensorflowNNInstance()

```
FilterManagementLibrary::TFIntegration::TensorflowNNInstance::TensorflowNNInstance (
            TensorflowNNModelDescription nnModelDescription,
            int numThreads = 1 )
```

Constructor of TensorflowNNInstance.

This is the only constructor the TensorflowNNInstance has, and the only one that should be allowed! As an object of type TensorflowNNInstance is useless without a TensorflowNNModelDescription, it is MANDATORY to have a constructor to which an object of that type is passed. Additionally, it needs to be specified how many threads Tensorflow is allowed to use for computation! An Tensor (of type tensorflow::Tensor), which is used as input Tensor, of type specified in the model description will be created (for now, only uint8_t and float are supported!) All other class members will be initialised to their default values.

Parameters

TensorflowNNModelDescription	nnModelDescription the model description containing all information needed to load and use the Tensorflow model.
int	numThreads number of threads Tensorflow is allowed to use.

References FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::channels, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_floating, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputHeight, inputTensor, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputWidth, and setupModelFromFile().

7.19.3.2 ∼TensorflowNNInstance()

```
FilterManagementLibrary::TFIntegration::TensorflowNNInstance::∼TensorflowNNInstance ( )
```

Destructor of the TensorflowNNInstance class.

Destructor whose only task is to close the Tensorflow Session which was used for this instance.

References tensorflowSession.

7.19.4 Member Function Documentation

7.19.4.1 adjustModelFile()

```
void FilterManagementLibrary::TFIntegration::TensorflowNNInstance::adjustModelFile (
             std::string modelFile )
```

Allows to change ONLY the model file path set in the model description.

Can be used to adjust the path of the model file of the TensorflowNNModelDescription at run-time.

Parameters

std::string	new path to the modelFile to use.

Returns

void

References FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::modelFile, and nnModelDescription.

7.19.4.2 applyImageInputVector()

```
bool FilterManagementLibrary::TFIntegration::TensorflowNNInstance::applyImageInputVector (
            uint8_t *** inputVector,
            int imageHeight,
            int imageWidth,
            int channels )
```

Applies an image vector to the instance of the Tensorflow model.

This takes a 3D uint8_t vector and will set the data on the input Tensor of the model. It is intended to be used for 3 Channel RGB images. If the model expects float as input type instead, the uint8_t values will be converted accordingly using the mean and std value provided with the model description.

Parameters

*uint8_t*** *	image 3D array of uint8_t values which will be applied to the model
int	height height of the image (columns of the vector)
int	width width of the image (rows of the vector)
int	channels channels of the image (depth of the vector)

Returns

bool true if the vector was successfully applied to the model, false otherwise

References applyInputVectorFromFloatData(), ERROR_INPUT_SIZE_MISMATCH, i, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_floating, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_mean, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_std, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputHeight, inputTensor, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputWidth, lastError, and nnModelDescription.

7.19.4.3 applyInputVectorFromFloatData()

```
bool FilterManagementLibrary::TFIntegration::TensorflowNNInstance::applyInputVectorFromFloat←
Data (
            float * floatData,
            const int heigth,
            const int width,
            const int channels )
```

Applies an already float input vector to the neuronal network model.

Used std::copy_n to directly copy the flat float vector to the input tensor May not always work (or at all).

References getLastError(), and inputTensor.

7.19.4.4 getInputTensor()

```
tensorflow::Tensor * FilterManagementLibrary::TFIntegration::TensorflowNNInstance::getInput↩
Tensor ( )
```

Returns the input Tensor of this instance.

Normally the Tensorflow input Tensor for the corresponding neuronal network model should'nt be visible outside this class. However it can be useful to provide a direct access to this Tensor, because specific tasks may need a specific way to apply an input to the network. TensorflowOpenCVUtils, for example, use this function for the fastApplyCVMatOnInputTensor functions. This, only use this function if you know what you're doing!

References inputTensor, and ∼TensorflowNNInstance().

7.19.4.5 getLastError()

```
FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ErrorType FilterManagement↩
Library::TFIntegration::TensorflowNNInstance::getLastError ( ) const
```

Returns the last error that happened.

Returns

TensorflowNNInstance::ErrorType the enum value of the last error that happened when executing any previous function of TensorflowNNInstance

References getOutputTensors(), and lastError.

7.19.4.6 getModelDescription()

```
const FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription * FilterManagement↩
Library::TFIntegration::TensorflowNNInstance::getModelDescription ( ) const
```

Returns the model description which was passed to this instance.

Returns

TensorflowNNModelDescription the model description which was passed to this instance in constructor.

References getInputTensor(), and nnModelDescription.

7.19.4.7 getOutputTensors()

```
std::vector< tensorflow::Tensor > * FilterManagementLibrary::TFIntegration::TensorflowNN↩
Instance::getOutputTensors ( )
```

Returns the tensors containg the output of the previous inference.

Returns the output tensors of this instance. If an inference was successfully executed before, those tensors will hold the output of the neuronal network model used. However, to examine the output, getResultContainer() should be used.

Returns

std::vector<stensorflow::tensor>* vector containg the output Tensors

References outputTensors, and runInference().

7.19.4.8 getResultContainer()

```
FilterManagementLibrary::TFIntegration::TensorflowResultContainer FilterManagementLibrary::T↩
FIntegration::TensorflowNNInstance::getResultContainer ( )
```

Returns a container containing the result of the last inference.

Packs Tensorflow output tensors of this instance to a TensorflowNNResultContainer which can be used to process the result of the last inference.

Returns

TensorflowResultContainer container containting the result of the last inference.

References getModelDescription(), nnModelDescription, FilterManagementLibrary::TFIntegration::TensorflowNN↩ ModelDescription::outputLayerNames, and outputTensors.

7.19.4.9 runInference()

```
bool FilterManagementLibrary::TFIntegration::TensorflowNNInstance::runInference ( )
```

Let's the neuronal network examine the applied input.

An input vector should have been applied prior to calling this function. It will run the Tensorflow Session's run function which calculates the output to the given input. On failure, lastError will be set accordingly.

Returns

true if the inference was successfull, false otherwise

References adjustModelFile(), ERROR_SESSION_RUN_FAILED, inputLayerName, inputTensor, lastError, nn↩ ModelDescription, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::outputLayerNames, outputTensors, FilterManagementLibrary::Logger::printfln(), and tensorflowSession.

7.19.4.10 setupModelFromAssets()

```
bool FilterManagementLibrary::TFIntegration::TensorflowNNInstance::setupModelFromAssets (
            AAssetManager *const assetManager )
```

Sets up the Tensorflow model using the provieded model description.

In contrast to setupModelFromFile(), this functions interpretes the modelFile specified in the TensorflowNNModelDescriotion as an Android Assets file and thus tries to load the model from the Assets belonging to the App. Of course, this is only available under Android environments. If the file could be loaded, a graph is built from it and a new Tensorflow Session will be created. On failure, lastError will be set accordingly. if the model was loaded and setup successfully, false otherwise

References applyImageInputVector(), ERROR_COULD_NOT_ADD_GRAPH_TO_SESSION, ERROR_FAILED_TO_CONSTRUCT_NEW_SESSION, graphDef, inputLayerName, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputLayerNameStr, lastError, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::modelFile, nnModelDescription, FilterManagementLibrary::Logger::println(), FilterManagementLibrary::TensorflowAndroidJNIUtils::ReadFileToProto(), sessionOptions, and tensorflowSession.

7.19.4.11 setupModelFromFile()

```
bool FilterManagementLibrary::TFIntegration::TensorflowNNInstance::setupModelFromFile ( )
```

Sets up the Tensorflow model using the provieded model description.

Tries to load the Tensorflow model from the file specified in the TensorflowNNModelDescription which is passed to the constructor. If the file could be loaded, a graph is built from it and a new Tensorflow Session will be created. On failure, lastError will be set accordingly. if the model was loaded and setup successfully, false otherwise

References ERROR_COULD_NOT_ADD_GRAPH_TO_SESSION, ERROR_COULD_NOT_LOAD_MODEL, ERROR_FAILED_TO_CONSTRUCT_NEW_SESSION, ERROR_INVALID_MODEL_FILE, FilterManagementLibrary::Utilities::fileExists(), graphDef, inputLayerName, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputLayerNameStr, lastError, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::modelFile, nnModelDescription, FilterManagementLibrary::Logger::println(), sessionOptions, setupModelFromAssets(), and tensorflowSession.

7.19.5 Member Data Documentation

7.19.5.1 graphDef

```
tensorflow::GraphDef FilterManagementLibrary::TFIntegration::TensorflowNNInstance::graphDef
[protected]
```

Tensorflow graph (built from the model)

7.19.5.2 inputLayerName

```
std::unique_ptr<std::string> FilterManagementLibrary::TFIntegration::TensorflowNNInstance←
::inputLayerName [protected]
```

Name of the input layer built from NeuronalNetwork

7.19.5.3 inputTensor

```
tensorflow::Tensor FilterManagementLibrary::TFIntegration::TensorflowNNInstance::inputTensor
[protected]
```

Tensor where our input is copied to.

7.19.5.4 lastError

```
ErrorType FilterManagementLibrary::TFIntegration::TensorflowNNInstance::lastError [protected]
```

Error code of the last error that happened. See TensorflowNNInstance::ErrorType

7.19.5.5 nnModelDescription

```
TensorflowNNModelDescription FilterManagementLibrary::TFIntegration::TensorflowNNInstance←
::nnModelDescription [private]
```

Everything we need to load, build and run the model. See TensorflowNNModelDescription for a detailed description.

7.19.5.6 numThreads

```
int FilterManagementLibrary::TFIntegration::TensorflowNNInstance::numThreads = 1 [private]
```

Number of threads Tensorflow is allowed to use. Standard value is 1, but can be set via constructor.

7.19.5.7 outputTensors

```
std::vector<tensorflow::Tensor> FilterManagementLibrary::TFIntegration::TensorflowNNInstance←
::outputTensors [protected]
```

Stores the result of the inference

7.19.5.8 sessionOptions

```
tensorflow::SessionOptions FilterManagementLibrary::TFIntegration::TensorflowNNInstance←
::sessionOptions [protected]
```

Options for the session holding the model

7.19.5.9 tensorflowSession

```
std::unique_ptr<tensorflow::Session> FilterManagementLibrary::TFIntegration::TensorflowNN↩
Instance::tensorflowSession  [protected]
```

Stores and holds the instance of the used model

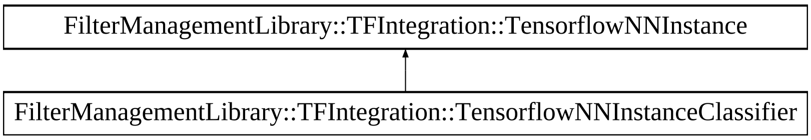
The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/TensorflowIntegration/TensorflowNNInstance.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/TensorflowIntegration/TensorflowNNInstance.cpp

7.20 FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier Class Reference

```
#include <TensorflowNNInstanceClassifier.h>
```

Inheritance diagram for FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier:


FilterManagementLibrary::TFIntegration::TensorflowNNInstance
FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier

Public Member Functions

- bool readLabelsFile (std::string path)

  Reads labels from a file.
- bool getTopNPredictions (const int numPredictions, const float threshold, std::vector< std::pair< float, int > > ∗outputList)

  Return the top N predictions of the network and their confidences.
- bool getTopNPredictionsStringVector (const int numPredictions, const float threshold, std::vector< std::string > ∗outputList)

  Return the top N labels of the network and their confidences.
- TensorflowNNInstanceClassifier (TensorflowNNModelDescription nnModelDescription, int numThreads=1)

  Constructor of TensorflowNNInstanceClassifier.

Private Attributes

- std::vector< std::string > loadedLabels
- int labelCount

Additional Inherited Members

7.20.1 Detailed Description

This is just a very basic class to fastly implement classification models like MobilenetV1/2 or Inception(V3) as they use the same method for output encoding (check getTopNPredictions(...) for more). It is not used in any of the productive PipeSystem Code, so see it more as an example or class for debugging and testing.

It only needs to be provided a correct label file for the corresponding trained model and a correct model description. Afterwards, getTopNPredictions(...) and getTopNPredictionsStringVector(...) can be used for easy output interpretation (don't forget to call runInference() first though).

7.20.2 Constructor & Destructor Documentation

7.20.2.1 TensorflowNNInstanceClassifier()

```
FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier::TensorflowNNInstance←-
Classifier (
             TensorflowNNModelDescription nnModelDescription,
             int numThreads = 1 )
```

Constructor of TensorflowNNInstanceClassifier.

Basically just calls the TensorflowNNInstance super constructor, which you may refer to for a more detailed description.

See also

Parameters

TensorflowNNModelDescription	nnModelDescription the model description containing all information needed to load and use the Tensorflow model.
int	numThreads number of threads Tensorflow is allowed to use.

References readLabelsFile().

7.20.3 Member Function Documentation

7.20.3.1 getTopNPredictions()

```
bool FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier::getTopNPredictions
(
             const int numPredictions,
             const float threshold,
             std::vector< std::pair< float, int > > * outputList )
```

Return the top N predictions of the network and their confidences.

Returns the top N confidence values over threshold in the provided vector, sorted by confidence in descending order.

Parameters

const	int numPredictions number of predictions that the outputList shall contain (the 'N' in getTopNPredictions(...))
const	float threshold min confidence value a prediction needs to have to be included in the outputList
std::vector<std::pair<float,int>	>∗ outputList pointer to a vector where the results will be written to.

References getTopNPredictionsStringVector(), i, and FilterManagementLibrary::TFIntegration::TensorflowNN↩ Instance::outputTensors.

7.20.3.2 getTopNPredictionsStringVector()

```
bool FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier::getTopNPredictions↩
StringVector (
            const int numPredictions,
            const float threshold,
            std::vector< std::string > * outputList )
```

Return the top N labels of the network and their confidences.

Uses getTopNPredictions(...) to get a list of the top N confidence predictions and their confidence values, and then uses the ID's to get the corresponding label tag from the labels file which should have been loaded prior to calling this function.

Parameters

const	int numPredictions number of predictions that the outputList shall contain (the 'N' in getTopNPredictions(...))
const	float threshold min confidence value a prediction needs to have to be included in the outputList
std::vector<std::string>∗	outputList pointer to a vector where the labels will be written to.

References getTopNPredictions(), labelCount, and loadedLabels.

7.20.3.3 readLabelsFile()

```
bool FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier::readLabelsFile (
            std::string path )
```

Reads labels from a file.

The specified file will be read line by line whereas each line represents an entry in the loadedLabels vector.

Parameters

std::string	path path to the labels file.

Returns

bool true if the labels file exists and could be read, false otherwise

References FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_TFNN_CLASSIFIER_INVALID_LABELS_FILE, getTopNPredictions(), labelCount, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::lastError, and loadedLabels.

7.20.4 Member Data Documentation

7.20.4.1 labelCount

```
int FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier::labelCount  [private]
```

Number of labels that were read. Assume these are equal to the output size of the used model!

7.20.4.2 loadedLabels

```
std::vector<std::string> FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier↵
::loadedLabels  [private]
```

List of labels loaded from file

The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/TensorflowIntegration/TensorflowNNInstanceClassifier.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/TensorflowIntegration/TensorflowNNInstanceClassifier.cpp

7.21 FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription Struct Reference

```
#include <TensorflowNNModelDescription.h>
```

Public Attributes

- bool input_floating = false
- float input_mean = 127.5f
- float input_std = 127.5f
- int inputWidth
- int inputHeight
- int channels = 3
- std::string modelFile
- std::string inputLayerNameStr
- std::vector< std::string > outputLayerNames

7.21.1 Member Data Documentation

7.21.1.1 channels

```
int FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::channels = 3
```

Amount of channels of the input layer (channels(RGB) = 3) Although if it may be flattened, Tensorflow provides functions to map tensors to access the data using x, y and z coordinates, which is what is used in TensorflowNNInstance::applyImageInputVector(...)

7.21.1.2 input_floating

```
bool FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_floating =
false
```

Specifys whether the input type of the neuronal network model is tensorflow::DT_UINT8 or tensorflow::DT_FLOAT (true means float, false means uint8). Others are not supported for now.

7.21.1.3 input_mean

```
float FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_mean = 127.5f
```

Only used if input_floating = true. As the default operations of the FilterManagementLibrary are implemented to use uint8_t, we need a way to convert those values to their float counterparts. For this, we need a mean and a std value for normalization. Formula is: float_value = (uint8_t_value - input_mean) / input_std;

7.21.1.4 input_std

```
float FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_std = 127.5f
```

Only used if input_floating = true. As the default operations of the FilterManagementLibrary are implemented to use uint8_t, we need a way to convert those values to their float counterparts. For this, we need a mean and a std value for normalization. Formula is: float_value = (uint8_t_value - input_mean) / input_std;

7.21.1.5 inputHeight

```
int FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputHeight
```

Height of the input layer. Although if it may be flattened, Tensorflow provides functions to map tensors to access the data using x, y and z coordinates, which is what is used in TensorflowNNInstance::applyImageInputVector(...)

7.21.1.6 inputLayerNameStr

```
std::string FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputLayer←
NameStr
```

Name of the input layer of the Neuronal Network model.

7.21.1.7 inputWidth

```
int FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputWidth
```

Width of the input layer. Although if it may be flattened, Tensorflow provides functions to map tensors to access the data using x, y and z coordinates, which is what is used in TensorflowNNInstance::applyImageInputVector(...)

7.21.1.8 modelFile

```
std::string FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::modelFile
```

Path to the protobuf model file which contains the model and where the Tensorflow graph is built from.

7.21.1.9 outputLayerNames

```
std::vector<std::string> FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription←
::outputLayerNames
```

List of names of the output layers that shall be used during inference. For output interpretation, you can also use these names to get the corresponding output tensor from a TensorflowResultContainer. But if your network model has really silly or annoying 1042 character long names, be aware that you could also use an ID tou get the corresponding tensor form TensorflowResultContainer. The ID corresponds to the position of the layer in this vector, so be sure to remember it!

The documentation for this struct was generated from the following file:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/TensorflowIntegration/Tensorflow←NNModelDescription.h

7.22 FilterManagementLibrary::TensorflowOpenCVUtils Class Reference

```
#include <TensorflowOpenCVUtils.h>
```

Static Public Member Functions

- static void fastApplyCVMatOnInputTensorUInt8 (cv::Mat ∗mat, tensorflow::Tensor ∗inputTensor)

  Applies the raw data of a openCV mat to an Tensorflow input tensor.
- static void fastApplyCVMatOnInputTensorFloat (cv::Mat ∗mat, tensorflow::Tensor ∗inputTensor, float mean, float std)

  Applies the raw data of a openCV mat to an Tensorflow input tensor.

7.22.1 Detailed Description

Some utilities to efficiently use OpenCV with the Filter management library. Remember YOU DON'T NEED THIS TO USE THE FILTER MANAGEMENT LIBRARY!!

7.22.2 Member Function Documentation

7.22.2.1 fastApplyCVMatOnInputTensorFloat()

```
void FilterManagementLibrary::TensorflowOpenCVUtils::fastApplyCVMatOnInputTensorFloat (
            cv::Mat ∗ mat,
            tensorflow::Tensor ∗ inputTensor,
            float mean,
            float std )  [static]
```

Applies the raw data of a openCV mat to an Tensorflow input tensor.

Does the same as fastApplyCVMatOnInputTensorUInt8, but additionally converts the uint8_t values to their corresponding float values using a mean and std value.

If you use this, make sure your CV Mat is in the right format, e.g. 8UC3. We could use a 32FC3 CV Float Mat for this, but we may wan't to apply our own std and mean values (don't know maybe we don't want to use 127.5f or vice versa), that's why we expect a normal uint based BGR Matrix). Also make sure mat and inputTensor have the same dimension!! We don't do any safety checks here, because we want to be *fast*

You may check TFNNBasedPipeFilter class description for how to use this function easily in a TFNNBasedPipeFilter (it's a one liner basically!).

Parameters

cv::Mat	∗mat pointer to the OpenCV matrix from which the data will be copied from
tensorflow::Tensor	∗inputTensor Tensorflow Tensor where the data will be copied to.

7.22.2.2 fastApplyCVMatOnInputTensorUInt8()

```
void FilterManagementLibrary::TensorflowOpenCVUtils::fastApplyCVMatOnInputTensorUInt8 (
            cv::Mat ∗ mat,
            tensorflow::Tensor ∗ inputTensor )  [static]
```

Applies the raw data of a openCV mat to an Tensorflow input tensor.

Uses OpenCV's forEach<...>() function to *fastly* iterate over all pixels of the OpenCV Mat and copy them to the input tensor. As forEach<...>() uses multi threading, this method is *WAY* faster (5 - 10 times in my tests) than naive per pixel copying.

If you use this, make sure your CV Mat is in the right format, e.g. 8UC3. Also make sure mat and inputTensor have the same dimension!! We don't do any safety checks here, because we want to be *fast* (although it's not called "fast" because it lacks safety, I want to notice).

You may check TFNNBasedPipeFilter class description for how to use this function easily in a TFNNBasedPipeFilter (it's a one liner basically!).

Parameters

cv::Mat	∗mat pointer to the OpenCV matrix from which the data will be copied from
tensorflow::Tensor	∗inputTensor Tensorflow Tensor where the data will be copied to.

References fastApplyCVMatOnInputTensorFloat().

The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/TensorflowOpenCVUtils.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/TensorflowOpenCVUtils.cpp

7.23 FilterManagementLibrary::TFIntegration::TensorflowResultContainer Class Reference

```
#include <TensorflowResultContainer.h>
```

Public Member Functions

- TensorflowResultContainer (std::vector< tensorflow::Tensor > ∗outputTensors, std::vector< std::string > outputTensorNames)

  Constructor of TensorflowResultContainer All information / data contained in an objecct of type TensorflowResultContainer will be set using this constructor. It does not contain any other class member variables!
- tensorflow::Tensor ∗ getOutputTensorByID (int id) const

  Returns a pointer to the output Tensor with the given ID.
- tensorflow::Tensor ∗ getOutputTensorByLayerName (std::string layerName) const

  Returns a pointer to the output Tensor at the given layer name.

Private Attributes

- std::vector< tensorflow::Tensor > ∗ outputTensors
- std::vector< std::string > outputTensorNames

7.23.1 Detailed Description

This class is a container for the results of a TensorflowNNInstance. It wraps the output tensors of an inference and their corresponding names so the internal structurization of those tensors can be hidden. It allows to get the Tensor of interest by name, so one does not have to remember the ID (which is the order of the output layer names provided in the TensorflowNNModelDescription used to create the TensorflowNNInstance.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 TensorflowResultContainer()

```
FilterManagementLibrary::TFIntegration::TensorflowResultContainer::TensorflowResultContainer (
            std::vector< tensorflow::Tensor > * outputTensors,
            std::vector< std::string > outputTensorNames )
```

Constructor of TensorflowResultContainer All information / data contained in an objecct of type TensorflowResult↩ Container will be set using this constructor. It does not contain any other class member variables!

Parameters

std::vector<tensorflow::Tensor>∗	outputTensors the list of output tensors of the corresponding inference (the Tensors containing the result)
std::vector<std::string>	outputTensorNames layer names of the output tensors. The names *HAVE* to be in the same order as the outputTensors vector! Otherwise a wrong tensor would be returned when using the getOutTensorByLayerName(...) function.

References getOutputTensorByID().

7.23.3 Member Function Documentation

7.23.3.1 getOutputTensorByID()

```
tensorflow::Tensor * FilterManagementLibrary::TFIntegration::TensorflowResultContainer::get↩
OutputTensorByID (
            int id ) const
```

Returns a pointer to the output Tensor with the given ID.

Returns

tensorflow::Tensor the output Tensor with the given ID.

References getOutputTensorByLayerName(), and outputTensors.

7.23.3.2 getOutputTensorByLayerName()

```
tensorflow::Tensor * FilterManagementLibrary::TFIntegration::TensorflowResultContainer::get←
OutputTensorByLayerName (
             std::string layerName ) const
```

Returns a pointer to the output Tensor at the given layer name.

Returns

tensorflow::Tensor the output Tensor at the given layer name.

References outputTensorNames, and outputTensors.

7.23.4 Member Data Documentation

7.23.4.1 outputTensorNames

```
std::vector<std::string> FilterManagementLibrary::TFIntegration::TensorflowResultContainer←
::outputTensorNames [private]
```

List of output tensors provided by the TensorflowNNModelDescription of the corresponding TensorflowNNInstancce. The order of the tensor names here matches the order of the tensor names in the model description! So the ID of the first output layer name in the model description is also the ID of the corresponding output tensor in this vector.

7.23.4.2 outputTensors

```
std::vector<tensorflow::Tensor>* FilterManagementLibrary::TFIntegration::TensorflowResult←
Container::outputTensors [private]
```

List of output tensors created by TensorflowNNInstance according to the output layer names wich were provided in TensorflowNNModelDescription. TensorflowNNInstance assures that the order of output layer names provided matches the order of the output tensors. So the ID of the first output layer name in the model description is also the ID of the corresponding output tensor in this vector.

The documentation for this class was generated from the following files:

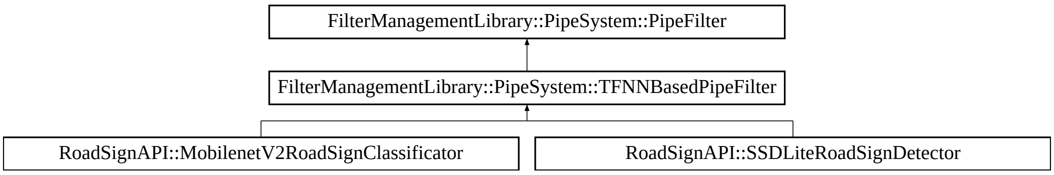
- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/TensorflowIntegration/Tensorflow←ResultContainer.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/TensorflowIntegration/Tensorflow←ResultContainer.cpp

7.24 FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter Class Reference

Abstract template class to derive Tensorflow based filters from.

```
#include <TFNNBasedPipeFilter.h>
```

Inheritance diagram for FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter:


FilterManagementLibrary::PipeSystem::PipeFilter
FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter
RoadSignAPI::MobilenetV2RoadSignClassificator
RoadSignAPI::SSDLiteRoadSignDetector

Public Member Functions

- TFNNBasedPipeFilter (TFIntegration::TensorflowNNModelDescription nnModelDescription, int numThreads)

  Standard constructor of TFNNBasedPipeFilter.
- void adjustModelFile (std::string modelFile)

  Allows to change ONLY the model file path set in the model description.
- bool setupModelFromFile ()

  Set's up the Tensorflow model of the TensorflowNNInstance.
- bool setupModelFromAssets (AAssetManager ∗const assetManager)

  Set's up the Tensorflow model of the TensorflowNNInstance.
- virtual ∼TFNNBasedPipeFilter ()

  Destructor of TFNNBasedPipeFilter. Does nothing by now.

Protected Member Functions

- bool applyNNImageInputVector (uint8_t ∗∗∗image, const int height, const int width, const int channels)

  Applies an image vector to the TensorflowNNInstance.
- bool evaluateInputVectorByNN ()

  Let's the neuronal network evaluate the given input.
- tensorflow::Tensor ∗ getNNInputTensor ()

  Returns the input tensor of the underlaying neuronal network.
- const FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription ∗ getNNModelDescription () const

  Returns the model description of the underlaying neuronal network.
- TFIntegration::TensorflowNNInstance::ErrorType getNeuralNetLastError () const

  Returns the last error that happened concerning the neuronal instance.
- virtual void onNNEvaluationFinished (const TFIntegration::TensorflowResultContainer resultContainer)=0

Private Attributes

- TFIntegration::TensorflowNNInstance tfNNInstance

Additional Inherited Members

7.24.1 Detailed Description

Abstract template class to derive Tensorflow based filters from.

A TFNNBasedPipeFilter is a special type of filter (thus derived from PipeFilter) already providing some functions to easily implemented filters based on Tensorflow Neuronal Networks. Strictly speaking the main processing of these filters is done by a preloaded Tensorflow model. The filter may take a part of the working data set and feed it into the model using the output of it's inference to generate the result of this filter. If you use it, make sure to call evaluateInputVectorByNN() in order to let the neuronal network do it's predictions.

For this, the TFNNBasedPipeFilter abstract class uses a TensorflowNNInstance (TensorflowNNInstanceClassifier is not used, as this is just a test class for basic classification models with a predefined way of interpreting their outputs, which is not assured to be a general way to interprete the output other classificator networks may use).

It provides a predefined easy way to apply input vectors from a 3 dimensional array of uint8_t's (applyNNImageInputVector). If the underlying model expects floats however, the conversion will be done automatically (see TensorflowNNInstance for more details). If a more specific way to apply an input is needed, the function getInputTensor() is provided which provides a direct access to the input tensor of the Neural Network Instance. Also keep in mind we provide some utility functions for easy OpenCV integration! In detail, offering some functions to directly apply a cv::Mat to an input tensor (see TensorflowOpenCVUtils).

A basic example to performantely effective apply a matrix to the underlying neuronal network, expecting uint8_t as input vector, from within a derivate of TFNNBasedPipeFilter would be: TensorflowOpenCVUtils::fastApplyCVMatOnInputTensorUInt8(&yourMat, this->getNNInputTensor()); simple as that!

However, keep in mind TensorflowOpenCVUtils do NOT have to be compiled with and linked to a exectuable or libary using the FilterManagementLibrary, if for example you don't use OpenCV (Mat's as input methods).

7.24.2 Constructor & Destructor Documentation

7.24.2.1 TFNNBasedPipeFilter()

```
FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::TFNNBasedPipeFilter (
             TFIntegration::TensorflowNNModelDescription nnModelDescription,
             int numThreads )
```

Standard constructor of TFNNBasedPipeFilter.

The TFNNBasedPipeFilter will use a TensorflowNNInstance to load and instantiate Tensorflow model (and a corresponding session). For this, it needs to tell the TensorflowNNInstance a description of a model to use (TensorflowNNModelDescription) and how many threads Tensorflow is allowed to use for computation.

Parameters

TensorflowNNModelDescription	nnModelDescription a description of the Tensorflow model which will be used by the filter.
int	numThreads number of threads Tensorflow is allowed to use for computation.

References applyNNImageInputVector().

7.24.2.2 ∼TFNNBasedPipeFilter()

```
FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::∼TFNNBasedPipeFilter ( )  [virtual]
```

Destructor of TFNNBasedPipeFilter. Does nothing by now.

7.24.3 Member Function Documentation

7.24.3.1 adjustModelFile()

```
void FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::adjustModelFile (
             std::string modelFile )
```

Allows to change ONLY the model file path set in the model description.

Can be used to adjust the path of the model file of the TensorflowNNModelDescription at run-time. TensorflowNNInstance's adjustModelFile(...) function will be called accordingly.

Parameters

std::string	new path to the modelFile to use.

Returns

void

References FilterManagementLibrary::TFIntegration::TensorflowNNInstance::adjustModelFile(), getNeuralNetLastError(), and tfNNInstance.

7.24.3.2 applyNNImageInputVector()

```
bool FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::applyNNImageInputVector (
             uint8_t *** image,
             const int height,
             const int width,
             const int channels )  [protected]
```

Applies an image vector to the TensorflowNNInstance.

This takes a 3D uint8_t vector and will call the TensorflowNNInstance's applyImageInputVector(...)function accordingly, which will set the data on the input Tensor of the model. It originally was intended to be used for 3 Channel RGB images, as the TensorflowNNInstance's function may scale the input as needed (i.e. normalization using floats and a mean and std value).

Parameters

*uint8_t****	image 3D array of uint8_t values which will be applied to the model
int	height height of the image (columns of the vector)
int	width width of the image (rows of the vector)
int	channels channels of the image (depth of the vector)

Returns

> bool true if the vector was successfully applied to the model, false otherwise

References evaluateInputVectorByNN(), and tfNNInstance.

7.24.3.3 evaluateInputVectorByNN()

```
bool FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::evaluateInputVectorByNN ( )
[protected]
```

Let's the neuronal network evaluate the given input.

An input vector should have been applied to the neuronal network (the TensorflowNNInstance) beforehand, by any means. This will call the TensorflowNNInstance's runInference() function accordingly and will call the onNNEvaluationFinished() callback function on success (synchronously).

Returns

> bool true if the inference (TensorflowNNInstance) was successfull, false otherwise.

References getNNInputTensor(), FilterManagementLibrary::TFIntegration::TensorflowNNInstance::getResultContainer(), onNNEvaluationFinished(), FilterManagementLibrary::Logger::printfln(), FilterManagementLibrary::TFIntegration::TensorflowNNInstance::runInference(), and tfNNInstance.

7.24.3.4 getNeuralNetLastError()

```
FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ErrorType FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::getNeuralNetLastError ( ) const [protected]
```

Returns the last error that happened concerning the neuronal instance.

Returns

> TensorflowNNInstance::ErrorType the enum value of the last error that happened concering the TensorflowNNInstance

References FilterManagementLibrary::TFIntegration::TensorflowNNInstance::getLastError(), tfNNInstance, and ∼TFNNBasedPipeFilter().

7.24.3.5 getNNInputTensor()

```
tensorflow::Tensor * FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::getNNInput↩
Tensor ( ) [protected]
```

Returns the input tensor of the underlaying neuronal network.

As we want to hide the object of TensorflowNNInstance from a derived filter, we provide this function to directly access the input tensor of the underlaying tensorflow neuronal network.

Returns

tensorflow::Tensor∗ pointer to the input tensor

References FilterManagementLibrary::TFIntegration::TensorflowNNInstance::getInputTensor(), getNNModelDescription(), and tfNNInstance.

7.24.3.6 getNNModelDescription()

```
const FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription * FilterManagement↩
Library::PipeSystem::TFNNBasedPipeFilter::getNNModelDescription ( ) const [protected]
```

Returns the model description of the underlaying neuronal network.

As we want to hide the object of TensorflowNNInstance from a derived filter, we provide this function to a const pointer to the model description of the underlaying neuronal network.

Returns

FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription∗ pointer to the model description.

References FilterManagementLibrary::TFIntegration::TensorflowNNInstance::getModelDescription(), setupModelFromFile(), and tfNNInstance.

7.24.3.7 onNNEvaluationFinished()

```
virtual void FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::onNNEvaluationFinished
(
             const TFIntegration::TensorflowResultContainer resultContainer ) [protected],
[pure virtual]
```

Implemented in RoadSignAPI::MobilenetV2RoadSignClassificator, and RoadSignAPI::SSDLiteRoadSignDetector.

7.24.3.8 setupModelFromAssets()

```
bool FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::setupModelFromAssets (
             AAssetManager *const assetManager )
```

Set's up the Tensorflow model of the TensorflowNNInstance.

CAUTION: THIS FUNCTION IS ONLY AVAILABLE UNDER ANDROID ENVIRONMENTS! This will call TensorflowNNInstance's setupModelFromAssets() function accordingly, which will try to load and build the Tensorflow model from the given TensorflowNNModelDescription (passed to constructor). In contrast to setupModelFromFile(), the modelFile given in the TensorflowNNModelDescription will be threated as an android assets file and loaded via an AssetsManager which needs to be passed to this function.

Parameters

const	AAssetManager∗ assetManager Pointer to an AssetManager which, by any means, needs to to be passed from the Java side of the code.

Returns

bool true if network model was setup successfully, false otherwise

References adjustModelFile(), FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_COULD_NOT_ADD_GRAPH_TO_SESSION, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_COULD_NOT_LOAD_MODEL, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_FAILED_TO_CONSTRUCT_NEW_SESSION, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_INVALID_MODEL_FILE, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::getLastError(), FilterManagementLibrary::Logger::printfln(), FilterManagementLibrary::TFIntegration::TensorflowNNInstance::setupModelFromAssets(), and tfNNInstance.

7.24.3.9 setupModelFromFile()

```
bool FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::setupModelFromFile ( )
```

Set's up the Tensorflow model of the TensorflowNNInstance.

This will call TensorflowNNInstance's setupModelFromFile() function accordingly, which will try to load and build the Tensorflow model from the given TensorflowNNModelDescription (passed to constructor)

Returns

bool true if network model was setup successfully, false otherwise

References FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_COULD_NOT_ADD_GRAPH_TO_SESSION, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_COULD_NOT_LOAD_MODEL, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_FAILED_TO_CONSTRUCT_NEW_SESSION, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_INVALID_MODEL_FILE, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::getLastError(), FilterManagementLibrary::Logger::printfln(), setupModelFromAssets(), FilterManagementLibrary::TFIntegration::TensorflowNNInstance::setupModelFromFile(), and tfNNInstance.

7.24.4 Member Data Documentation

7.24.4.1 tfNNInstance

```
TFIntegration::TensorflowNNInstance FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter::tfNNInstance [private]
```

The filter local instance of the underlying tensorflow neuronal network model.

The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/PipeSystem/TFNNBasedPipeFilter.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/PipeSystem/TFNNBasedPipeFilter.cpp

7.25 FilterManagementLibrary::Utilities Class Reference

Providing some basic utilities, like checking if a file exists.

```
#include <Utilities.h>
```

Static Public Member Functions

- static bool fileExists (const std::string &name)

  Check if a file with the given path exists.

7.25.1 Detailed Description

Providing some basic utilities, like checking if a file exists.

7.25.2 Member Function Documentation

7.25.2.1 fileExists()

```
bool FilterManagementLibrary::Utilities::fileExists (
             const std::string & name )  [static]
```

Check if a file with the given path exists.

Parameters

const	std::string& name string containing the path of the file

Returns

true if the file exists, false if the file does not exist or if we have insufficient permission to read it.

The documentation for this class was generated from the following files:

- RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/Utilities.h
- RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/Utilities.cpp

Chapter 8

File Documentation

8.1 RoadSignAPI_Standalone_Tensorflow/_10_introduction.dox File Reference

8.2 RoadSignAPI_Standalone_Tensorflow/codesnippets.cpp File Reference

Functions

- Mat scaledImage (settings.inputHeigth, settings.inputWidth, CV_32FC3)
- cv::Mat cameraImg (settings.inputHeigth, settings.inputWidth, CV_32FC3, p)
- scaledImage convertTo (cameraImg, CV_32FC3)

Variables

- OpenCV mat to tensorflow with CV FakeMat cv::Mat image = cv::imread("grace_hopper.bmp")
- gettimeofday & start_time
- int i = 0
- float $*$ p = tfInstance.getInputTensor()->flat<float>().data()

8.2.1 Function Documentation

8.2.1.1 cameraImg()

```
cv::Mat cameraImg (
            settings.  inputHeigth,
            settings.  inputWidth,
            CV_32FC3 ,
            p  )
```

8.2.1.2 convertTo()

```
scaledImage convertTo (
             cameraImg ,
             CV_32FC3  )
```

8.2.1.3 scaledImage()

```
Mat scaledImage (
             settings.  inputHeigth,
             settings.  inputWidth,
             CV_32FC3  )
```

8.2.2 Variable Documentation

8.2.2.1 i

```
int i = 0
```

8.2.2.2 image

```
OpenCV mat to tensorflow with CV FakeMat cv::Mat image = cv::imread("grace_hopper.bmp")
```

8.2.2.3 p

```
float* p = tfInstance.getInputTensor()->flat<float>().data()
```

8.2.2.4 start_time

```
gettimeofday& start_time
```

8.3 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/limiting↩_file_input_stream.h File Reference

Classes

- class tensorflow::android::LimitingFileInputStream

Namespaces

- tensorflow
- tensorflow::android

8.4 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/↩ Logger.h File Reference

Classes

- class FilterManagementLibrary::Logger

  Logger providing Desktop Linux <-> Android platform independent logging.

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.

8.5 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/Pipe↩ System/PipeFilter.h File Reference

Classes

- class FilterManagementLibrary::PipeSystem::PipeFilter

  Base class were all filters are derived from.

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.
- FilterManagementLibrary::PipeSystem

  Everything that has to do with the pipes and filters part of the library.

8.6 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/Pipe↩ System/PipeRegisteredFilters.h File Reference

Classes

- struct FilterManagementLibrary::PipeSystem::PipeRegisteredFilters

  Template struct.

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.
- FilterManagementLibrary::PipeSystem

  Everything that has to do with the pipes and filters part of the library.

8.7 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/Pipe↩ System/PipeWorkingDataSet.h File Reference

Classes

- struct FilterManagementLibrary::PipeSystem::PipeWorkingDataSet

  Template structfor PipeWorkingDataSets.

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.
- FilterManagementLibrary::PipeSystem

  Everything that has to do with the pipes and filters part of the library.

8.8 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/Pipe↩ System/ProcessingPipeline.h File Reference

Classes

- class FilterManagementLibrary::PipeSystem::ProcessingPipeline

  Pipeline of the pipes and filters architecture.

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.
- FilterManagementLibrary::PipeSystem

  Everything that has to do with the pipes and filters part of the library.

8.9 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/Pipe↩ System/TFNNBasedPipeFilter.h File Reference

Classes

- class FilterManagementLibrary::PipeSystem::TFNNBasedPipeFilter

  Abstract template class to derive Tensorflow based filters from.

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.
- FilterManagementLibrary::PipeSystem

  Everything that has to do with the pipes and filters part of the library.

8.10 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/↩ TensorflowAndroidJNIUtils.h File Reference

Classes

- class FilterManagementLibrary::TensorflowAndroidJNIUtils

Namespaces

- google
- google::protobuf
- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.

8.11 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/↩ TensorflowIntegration/TensorflowNNInstance.h File Reference

Classes

- class FilterManagementLibrary::TFIntegration::TensorflowNNInstance

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.
- FilterManagementLibrary::TFIntegration

8.12 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/↩ TensorflowIntegration/TensorflowNNInstanceClassifier.h File Reference

Classes

- class FilterManagementLibrary::TFIntegration::TensorflowNNInstanceClassifier

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.
- FilterManagementLibrary::TFIntegration

8.13 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/↩ TensorflowIntegration/TensorflowNNModelDescription.h File Reference

Classes

- struct FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.
- FilterManagementLibrary::TFIntegration

8.14 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/↩ TensorflowIntegration/TensorflowResultContainer.h File Reference

Classes

- class FilterManagementLibrary::TFIntegration::TensorflowResultContainer

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.
- FilterManagementLibrary::TFIntegration

8.15 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/↩ TensorflowOpenCVUtils.h File Reference

Classes

- class FilterManagementLibrary::TensorflowOpenCVUtils

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.

8.16 RoadSignAPI_Standalone_Tensorflow/header_files/FilterManagementLibrary/↩ Utilities.h File Reference

Classes

- class FilterManagementLibrary::Utilities

  Providing some basic utilities, like checking if a file exists.

Namespaces

- FilterManagementLibrary

  Everything the FilterManagementLibrary provides.

8.17 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/DetectedSign↩ Combination.h File Reference

Classes

- class RoadSignAPI::DetectedSignCombination

Namespaces

- RoadSignAPI

8.18 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/DetectedSign↩ Descriptor.h File Reference

Classes

- struct RoadSignAPI::DetectedSignDescriptor

Namespaces

- RoadSignAPI

8.19 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/Classified↩ SignsGrouper.h File Reference

Classes

- class RoadSignAPI::ClassifiedSignsGrouper

Namespaces

- RoadSignAPI

8.20 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/Detection↩ BasedImageSlicer.h File Reference

Classes

- class RoadSignAPI::DetectionBasedImageSlicer

Namespaces

- RoadSignAPI

8.21 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/Mobilenet↩ V2RoadSignClassificator.h File Reference

Classes

- class RoadSignAPI::MobilenetV2RoadSignClassificator

Namespaces

- RoadSignAPI

8.22 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/Road↩ SignDuplicationDeleter.h File Reference

Classes

- class RoadSignAPI::RoadSignDuplicationDeleter

Namespaces

- RoadSignAPI

8.23 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/Filters/SSDLite↩ RoadSignDetector.h File Reference

Classes

- class RoadSignAPI::SSDLiteRoadSignDetector

Namespaces

- RoadSignAPI

8.24 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/RoadSignAPI.h File Reference

Classes

- class RoadSignAPI::RoadSignAPI

Namespaces

- RoadSignAPI

8.25 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/RSAPIPipeRegisteredFilters/RSAPIPipeRegisteredFilters.h File Reference

Classes

- struct RoadSignAPI::RSAPIPipeRegisteredFilters

Namespaces

- RoadSignAPI

8.26 RoadSignAPI_Standalone_Tensorflow/header_files/RoadSignAPI/RSAPIWorkingDataSet/RSAPIWorkingDataSet.h File Reference

Classes

- struct RoadSignAPI::RSAPIWorkingDataSet

Namespaces

- RoadSignAPI

8.27 RoadSignAPI_Standalone_Tensorflow/main.cpp File Reference

Typedefs

- typedef Point3_< uint8_t > Pixel

Functions

- double get_us (struct timeval t)
- bool readLabelsFile (std::string path, std::vector< std::string > ∗output)
- bool openCVMatTouint8_tArray (cv::Mat ∗mat, uint8_t ∗output, int ∗width, int ∗heigth)
- bool openCVMatTouint8_t3DArray (cv::Mat ∗mat, uint8_t ∗∗∗output, int ∗width, int ∗heigth)
- bool applyFromCVMat (cv::Mat ∗mat, tensorflow::Tensor ∗inputTensor)
- std::string type2str (int type)
- int main ()

8.27.1 Typedef Documentation

8.27.1.1 Pixel

```
typedef Point3_<uint8_t> Pixel
```

8.27.2 Function Documentation

8.27.2.1 applyFromCVMat()

```
bool applyFromCVMat (
             cv::Mat * mat,
             tensorflow::Tensor * inputTensor )
```

References get_us().

8.27.2.2 get_us()

```
double get_us (
             struct timeval t )
```

8.27.2.3 main()

```
int main ( )
```

References RoadSignAPI::DetectedSignCombination::addDetectedSign(), RoadSignAPI::DetectedSignDescriptor::classifierApprovedClassID, RoadSignAPI::DetectedSignDescriptor::detectorConfidence, RoadSignAPI::DetectedSignCombination::getDetectedSignsAmount(), RoadSignAPI::DetectedSignCombination::getGestimatedPolePositionX(), RoadSignAPI::DetectedSignCombination::getSignsInCombination(), i, RoadSignAPI::DetectedSignDescriptor::lowerRight, readLabelsFile(), scaledImage(), RoadSignAPI::RoadSignAPI::staticFeedImage(), RoadSignAPI::RoadSignAPI::staticGetDetectedSignCombinations(), RoadSignAPI::RoadSignAPI::staticInit(), type2str(), and RoadSignAPI::DetectedSignDescriptor::upperLeft.

8.27.2.4 openCVMatTouint8_t3DArray()

```
bool openCVMatTouint8_t3DArray (
             cv::Mat * mat,
             uint8_t *** output,
             int * width,
             int * heigth )
```

8.27.2.5 openCVMatTouint8_tArray()

```
bool openCVMatTouint8_tArray (
             cv::Mat * mat,
             uint8_t * output,
             int * width,
             int * heigth )
```

8.27.2.6 readLabelsFile()

```
bool readLabelsFile (
             std::string path,
             std::vector< std::string > * output )
```

8.27.2.7 type2str()

```
std::string type2str (
             int type )
```

8.28 RoadSignAPI_Standalone_Tensorflow/mainold.cpp File Reference

Typedefs

- typedef Point3_< uint8_t > Pixel

Functions

- double get_us (struct timeval t)
- bool openCVMatTouint8_tArray (cv::Mat *mat, uint8_t *output, int *width, int *heigth)
- bool openCVMatTouint8_t3DArray (cv::Mat *mat, uint8_t ***output, int *width, int *heigth)
- std::string type2str (int type)
- static void GetTopN (const Eigen::TensorMap< Eigen::Tensor< float, 1, Eigen::RowMajor >, Eigen::Aligned > &prediction, const int num_results, const float threshold, std::vector< std::pair< float, int > > *top_results)
- void testNet ()
- int main ()

8.28.1 Typedef Documentation

8.28.1.1 Pixel

```
typedef Point3_<uint8_t> Pixel
```

8.28.2 Function Documentation

8.28.2.1 get_us()

```
double get_us (
            struct timeval t )
```

8.28.2.2 GetTopN()

```
static void GetTopN (
            const Eigen::TensorMap< Eigen::Tensor< float, 1, Eigen::RowMajor >, Eigen::↩
Aligned > & prediction,
            const int num_results,
            const float threshold,
            std::vector< std::pair< float, int > > * top_results )  [static]
```

References i.

8.28.2.3 main()

```
int main ( )
```

References FilterManagementLibrary::TFIntegration::TensorflowNNInstance::applyImageInputVector(), FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_COULD_NOT_ADD_GRAPH_TO_SESSION, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_COULD_NOT_LOAD_MODEL, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_FAILED_TO_CONSTRUCT_NEW_SESSION, FilterManagementLibrary::TFIntegration::TensorflowNNInstance::ERROR_INVALID_MODEL_FILE, get_us(), FilterManagementLibrary::TFIntegration::TensorflowNNInstance::getLastError(), i, image, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_floating, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_mean, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::input_std, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputLayerNameStr, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::inputWidth, FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::modelFile, openCVMatTouint8_t3DArray(), FilterManagementLibrary::TFIntegration::TensorflowNNModelDescription::outputLayerNames, FilterManagementLibrary::Logger::printfln(), FilterManagementLibrary::TFIntegration::TensorflowNNInstance::runInference(), scaledImage(), and type2str().

8.28.2.4 openCVMatTouint8_t3DArray()

```
bool openCVMatTouint8_t3DArray (
             cv::Mat * mat,
             uint8_t *** output,
             int * width,
             int * heigth )
```

8.28.2.5 openCVMatTouint8_tArray()

```
bool openCVMatTouint8_tArray (
             cv::Mat * mat,
             uint8_t * output,
             int * width,
             int * heigth )
```

8.28.2.6 testNet()

```
void testNet ( )
```

References GetTopN(), i, image, openCVMatTouint8_t3DArray(), scaledImage(), and start_time.

8.28.2.7 type2str()

```
std::string type2str (
             int type )
```

8.29 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/↩ Logger.cpp File Reference

8.30 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/Pipe↩ System/PipeFilter.cpp File Reference

8.31 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/Pipe↩ System/ProcessingPipeline.cpp File Reference

8.32 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/Pipe↩ System/TFNNBasedPipeFilter.cpp File Reference

8.33 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/↩ TensorflowAndroidJNIUtils.cpp File Reference

Classes

- class anonymous_namespace{TensorflowAndroidJNIUtils.cpp}::IfstreamInputStream

Namespaces

- anonymous_namespace{TensorflowAndroidJNIUtils.cpp}

Variables

- static const char ∗const ASSET_PREFIX = "file:///android_asset/"

8.33.1 Variable Documentation

8.33.1.1 ASSET_PREFIX

```
const char* const ASSET_PREFIX = "file:///android_asset/"  [static]
```

8.34 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/↩ TensorflowIntegration/TensorflowNNInstance.cpp File Reference

8.35 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/↩ TensorflowIntegration/TensorflowNNInstanceClassifier.cpp File Reference

8.36 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/↩ TensorflowIntegration/TensorflowResultContainer.cpp File Reference

8.37 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/↩ TensorflowOpenCVUtils.cpp File Reference

Typedefs

- typedef cv::Point3_< uint8_t > Pixel

8.37.1 Typedef Documentation

8.37.1.1 Pixel

```
typedef cv::Point3_<uint8_t> Pixel
```

8.38 RoadSignAPI_Standalone_Tensorflow/source_files/FilterManagementLibrary/↩Utilities.cpp File Reference

8.39 RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/DetectedSign↩Combination.cpp File Reference

8.40 RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/Classified↩SignsGrouper.cpp File Reference

8.41 RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/Detection↩BasedImageSlicer.cpp File Reference

8.42 RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/Mobilenet↩V2RoadSignClassificator.cpp File Reference

8.43 RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/Road↩SignDuplicationDeleter.cpp File Reference

8.44 RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/Filters/SSDLite↩RoadSignDetector.cpp File Reference

8.45 RoadSignAPI_Standalone_Tensorflow/source_files/RoadSignAPI/RoadSignAPI.cpp File Reference

Index