openCV_test

Generated by Doxygen 1.13.2

Chapter 1

Film Shot Type Classification

This project focuses on the automatic classification of cinematic shot types (such as **close-up**, **medium**, and **wide** shots) from video or image data. The classification is based on detecting and analyzing visual features — primarily faces and objects — using classical computer vision techniques (e.g., **Haar cascades**) and handcrafted feature extraction.

The motivation for this tool is to support tasks such as:

- **Film style analysis**
- **Shot distribution statistics**
- **Automated metadata generation** for video datasets

The application processes input frame-by-frame, extracts spatial and geometric features, classifies the shot type, and optionally evaluates the performance against a labeled dataset.

1.1 Tasks

- `main` – Program entry point
- `UserStructs` – Definitions of shared data types and enums

1.1.1 Miroslaw

- `~FeatureDetector` – Object detection using Haar cascades~
- `~FeatureProcessorAndClassifier` – Feature extraction and shot type classification logic~
-

1.1.2 Marek

- `~FilmStatisticEval` – Aggregates classification results across frames~
- `~TestDatasetEval` – Compares predictions with ground truth labels~
- `~FileLoader` – Abstract interface for loading images or video~
- `ResultDisplayer` – Displays or exports results

1.2 Notes

- Do not make deep copies, since we are aiming to work with video
- Do not use static varibles inside classes
- Respect dataflow and do not edit it without telling others

1.3 Final Project Report

Here is report structure derived from example project in moodle

1.3.1 1. Title Page - Marek

- **Project title**: Film Shot Type Classification
- **Course**: Computer Vision
- **Authors**: [Your Name(s)]
- **Submission Date**: [Insert date]

1.3.2 2. Introduction - Miroslaw

- Brief background on the problem domain (e.g., film analysis, automatic metadata generation)
- Motivation for choosing this topic
- Main objectives and intended use of the system

1.3.3 3. System Overview - Marek

- High-level description of the system pipeline
- Diagram or figure (optional)
- Brief description of each major module:
  - Input loading (image/video)
  - Feature detection
  - Feature extraction
  - Shot classification
  - Statistical analysis
  - Output/display module

1.3.4 4. Implementation Details - Everyone describes his part

- Technical breakdown of each module
- Algorithms and techniques used (e.g., Haar cascades, OpenCV features)
- Programming language and tools used
- Code structure and how different parts interact

1.3.5 5. Evaluation - Marek

1.3.5.1 Quantitative Evaluation

- Accuracy, confusion matrix, or class-wise performance
- Dataset size and statistics

1.3.5.2 Qualitative Evaluation

- Visual examples:
  - Input frame(s)
  - Detected objects
  - Predicted shot type
  - Statistical output (timeline, chart, etc.)

1.3.6 6. Dataset Description - Miroslaw

- Source of the dataset (test images, videos, ground truth)
- Label definitions (e.g., CLOSE_UP, MEDIUM, WIDE)
- Preprocessing steps applied
- Any augmentations or synthetic data used

1.3.7 7. Results Summary - Miroslaw

- General observations
- Performance comparison if multiple methods were tested
- Strengths and weaknesses of the current implementation

1.3.8 8. Individual Contributions

Team Member	Contribution	Hours
Member 1	e.g., feature detection, classifier module	
Member 2	e.g., evaluation, testing, visualization	

1.3.9 9. Conclusion - Marek

- Final thoughts and achievements
- Summary of results and project goals met
- Ideas for improvement or future extensions

1.3.10 10. References

- Articles, documentation, libraries used (e.g., OpenCV)
- Any relevant tutorials, academic papers, GitHub repos

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 ClassificationResult Struct Reference

Contains the result of shot type classification.
`#include <UserStructs.hpp>`

Public Attributes

- ShotType **predictedType** = ShotType::UNKNOWN

  Most probable shot type, redundant but efective.
- std::map< ShotType, double > **probabilities**

  Probability distribution across shot types.

5.1.1 Detailed Description

Contains the result of shot type classification.
Holds both the most likely predicted shot type and a probability distribution over all possible shot types.
Set probability of detected ShotType to 1 if there is not better metrics
The documentation for this struct was generated from the following file:

- include/UserStructs.hpp

5.2 DetectedFeature Struct Reference

Represents a single detected feature in an image, with label and bounding box.
`#include <UserStructs.hpp>`

Public Attributes

- std::string **label**

  Type or name of the detected object (e.g., "face")
- cv::Rect **boundingBox**

  Bounding box of the detected object in image coordinates.

5.2.1 Detailed Description

Represents a single detected feature in an image, with label and bounding box.
This structure is used to return information about each detected object, such as its class label and location in the image.
The documentation for this struct was generated from the following file:

- include/UserStructs.hpp

5.3 FilmStatistics Class Reference

Collects and analyzes shot type data across a sequence of video frames.
`#include <FilmStatisticEval.hpp>`

Public Member Functions

- **FilmStatistics** ()=default

  Default constructor (analyzes every frame).
- void addConfigurationStruct (FilmStatisticsEvalConfig const &cfg)

  Adds configuration parameters for the evaluation.
- void addFrameResult (const double timestampMs, const ClassificationResult &result)

  Adds the classification result for a single frame.
- void exportToCSV (const std::string &path) const

  Exports all collected statistics to a CSV file.
- void **printSummary** () const

  Prints a textual summary of the shot distribution to the console.

Private Member Functions

- void normalizeProbs (std::map< ShotType, double > &aggregated_probs)

  Normalizes the aggregated probabilities so they sum to 1.
- void aggregateSlidingWindowProbs (std::deque< ClassificationResult > &sliding_window, std::map< Shot↩ Type, double > &aggregated_probs)

  Aggregates probabilities from a sliding window of classification results.
- void findShotTypeMaxProb (std::map< ShotType, double > &aggregated_probs, ClassificationResult &sample)

  Finds the shot type with the maximum probability from aggregated probabilities and updates the sample.
- void **oversampleInputData** ()

  Performs oversampling on input data to improve statistical robustness.
- void **computeEntropy** ()

  Computes the entropy of the current sliding window of classification results.
- void **computeEntropyVariance** ()

  Computes the variance of the entropy values over the sliding window.
- void **outputStatistics** ()

  Outputs collected statistics to the console or log.
- void appendSampleToTimeline (std::vector< std::pair< double, std::map< ShotType, double > > > &timeline)

  Appends the current sample's aggregated probabilities and timestamp to a timeline.
- void appendSampleToTimeline (std::vector< std::pair< double, ShotType > > &timeline)

  Appends the current sample's shot type and timestamp to a timeline.
- void appendEntropyToTimeline (std::vector< std::pair< double, double > > &timeline)

  Appends the current entropy value and timestamp to a timeline.
- void appendEntropyVarianceToTimeline (std::vector< std::pair< double, double > > &timeline)

  Appends the current entropy variance value and timestamp to a timeline.
- void appendCutDetectionToTimeline (std::vector< std::pair< double, bool > > &timeline)

  Appends the current cut detection flag and timestamp to a timeline.

Private Attributes

- std::map< ShotType, int > **shot_counts**

  Count of each shot type encountered in the video.
- std::vector< std::pair< double, std::map< ShotType, double > > > **prob_timeline**

  Timeline of aggregated shot type probabilities at given timestamps. Each entry maps a timestamp (double) to a map of ShotType and their probabilities.
- std::vector< std::pair< double, ShotType > > **shot_type_timeline**

  Timeline of detected shot types at given timestamps. Each entry maps a timestamp (double) to the detected ShotType.
- std::vector< std::pair< double, double > > **enthropy_timeline**

  Timeline of entropy values computed over a sliding window of samples. Each entry maps a timestamp (double) to the entropy value.
- std::vector< std::pair< double, double > > **enthropy_variance_timeline**

  Timeline of entropy variance values computed over a sliding window. Each entry maps a timestamp (double) to the entropy variance value.
- std::vector< std::pair< double, bool > > **cut_detection_timeline**

  Timeline of cut detection flags at given timestamps. Each entry maps a timestamp (double) to a boolean indicating if a cut was detected.
- std::deque< ClassificationResult > **oversampling_sliding_window**

  Sliding window storing classification results for oversampling.
- std::deque< ClassificationResult > **entropy_sliding_window**

  Sliding window storing classification results for entropy computation.
- std::deque< double > **entropy_variance_sliding_window**

  Sliding window storing entropy variance values.
- int **total_frames** = 0

  Total number of frames processed so far.
- int **evaluated_frames** = 0

  Number of frames that have been evaluated and included in statistics.
- ClassificationResult **current_sample_oversampled**

  Current oversampled classification result being processed.
- double **entropy** = 0.0

  Current entropy value computed from the sliding window.
- double **timestamp_ms** = 0.0

  Timestamp in milliseconds of the current frame/sample.
- double **entropy_variance** = 0.0

  Current entropy variance value computed from the sliding window.
- FilmStatisticsEvalConfig **config**

  Configuration parameters for the FilmStatistics evaluation.
- int **start_delay** = 0

  Number of initial frames to delay before starting analysis.
- double **frame_time_measurement**

  Duration of a single frame in milliseconds used for time measurement.

5.3.1 Detailed Description

Collects and analyzes shot type data across a sequence of video frames.
The FilmStatistics class is responsible for aggregating classification results from multiple frames and producing summary statistics about shot types used in a video. It also supports time-based analysis and exporting results.
This class is typically used at the end of a processing pipeline, after each frame has been classified. It stores both a timeline and cumulative counts of different shot types.
The analysis can be configured to skip frames using a configurable `step`, which allows subsampling of the video.
Example usage:

```
FilmStatisticsEvalConfig cfg;

FilmStatistics stats;
stats.addConfigurationStruct(cfg);
stats.addFrameResult(timestamp, result);
stats.printSummary();
stats.exportToCSV("shots.csv");
```

Author

Marek Tatýrek

Date

2025

See also

ClassificationResult

ShotType

5.3.2 Member Function Documentation

5.3.2.1 addConfigurationStruct()

```
void FilmStatistics::addConfigurationStruct (
            FilmStatisticsEvalConfig const & cfg)
```

Adds configuration parameters for the evaluation.

Parameters

cfg	Configuration struct containing evaluation parameters.

5.3.2.2 addFrameResult()

```
void FilmStatistics::addFrameResult (
            const double timestampMs,
            const ClassificationResult & result)
```

Adds the classification result for a single frame.

Parameters

timestampMs	Timestamp of the frame in milliseconds.
result	Classification result containing shot type.

5.3.2.3 aggregateSlidingWindowProbs()

```
void FilmStatistics::aggregateSlidingWindowProbs (
            std::deque< ClassificationResult > & sliding_window,
            std::map< ShotType, double > & aggregated_probs)  [private]
```

Aggregates probabilities from a sliding window of classification results.

Parameters

sliding_window	Deque containing classification results to aggregate.
aggregated_probs	Output map of ShotType to aggregated probability values.

5.3.2.4 appendCutDetectionToTimeline()

```
void FilmStatistics::appendCutDetectionToTimeline (
            std::vector< std::pair< double, bool > > & timeline)  [private]
```

Appends the current cut detection flag and timestamp to a timeline.

Parameters

timeline	Vector to append the timestamp and cut detection boolean.

5.3.2.5 appendEntropyToTimeline()

```
void FilmStatistics::appendEntropyToTimeline (
            std::vector< std::pair< double, double > > & timeline)  [private]
```

Appends the current entropy value and timestamp to a timeline.

Parameters

timeline	Vector to append the timestamp and entropy value.

5.3.2.6 appendEntropyVarianceToTimeline()

```
void FilmStatistics::appendEntropyVarianceToTimeline (
            std::vector< std::pair< double, double > > & timeline)  [private]
```

Appends the current entropy variance value and timestamp to a timeline.

Parameters

timeline	Vector to append the timestamp and entropy variance value.

5.3.2.7 appendSampleToTimeline() [1/2]

```
void FilmStatistics::appendSampleToTimeline (
            std::vector< std::pair< double, ShotType > > & timeline)  [private]
```

Appends the current sample's shot type and timestamp to a timeline.

Parameters

timeline	Vector to append the timestamp and shot type.

5.3.2.8 appendSampleToTimeline() [2/2]

```
void FilmStatistics::appendSampleToTimeline (
            std::vector< std::pair< double, std::map< ShotType, double > > > & timeline)
[private]
```

Appends the current sample's aggregated probabilities and timestamp to a timeline.

Parameters

timeline	Vector to append the timestamp and aggregated shot type probabilities.

5.3.2.9 exportToCSV()

```
void FilmStatistics::exportToCSV (
            const std::string & path) const
```

Exports all collected statistics to a CSV file.
The CSV includes both timeline data and summary shot counts.

Parameters

path	File path to export the data to.

5.3.2.10 findShotTypeMaxProb()

```
void FilmStatistics::findShotTypeMaxProb (
            std::map< ShotType, double > & aggregated_probs,
            ClassificationResult & sample)  [private]
```

Finds the shot type with the maximum probability from aggregated probabilities and updates the sample.

Parameters

aggregated_probs	Map of ShotType to their aggregated probabilities.
sample	Classification result to update with the shot type having max probability.

5.3.2.11 normalizeProbs()

```
void FilmStatistics::normalizeProbs (
            std::map< ShotType, double > & aggregated_probs)  [private]
```

Normalizes the aggregated probabilities so they sum to 1.

Parameters

aggregated_probs	Map of ShotType to their aggregated probabilities to normalize.

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

- include/FilmStatisticEval.hpp
- src/FilmStatisticEval.cpp

5.4 FilmStatisticsEvalConfig Struct Reference

Configuration structure for controlling statistical evaluation of film analysis.
`#include <UserStructs.hpp>`

Public Attributes

- size_t **input_step** = 1

  Frame step size – how many frames to skip during analysis (e.g., 1 = every frame, 2 = every second frame)
- size_t **input_oversample** = 1

  Number of frames to buffer for oversampling before producing aggregate result.
- size_t **entropy_window_size** = 20

  Window size for calculating mean entropy over time.
- size_t **entropy_variance_window_size** = 30

  Window size for entropy variance calculation (0 = disabled)
- double **cut_detect_entropy_threshold** = 1

  Threshold value for absolute entropy to trigger cut detection.
- bool **output_ratios_series** = false

  Whether to output ratio data (e.g., object area / frame area)
- size_t **output_shot_type_time_series**
- size_t **output_prob_time_series** = 1

  Interval for outputting classification probability time series (0 = disabled)

- size_t **output_entropy_time_series** = 1

  Interval for exporting entropy over time (0 = disabled)
- size_t **output_entropy_variance_time_series** = 1

  Interval for exporting entropy variance over time (0 = disabled)
- size_t **output_cut_detection_time_series** = 1

  Interval for exporting detected cuts (0 = disabled)
- double **cut_detection_entropy_treshold** = 1.3

  Absolute entropy value required to consider cut (used as hard trigger)
- size_t **cut_detection_history_window_size** = 15

  Number of previous frames to consider for cut decision history.
- double **cut_detection_entropy_diff_treshold** = 0.3

  Required difference in entropy between frames to confirm a cut.

5.4.1 Detailed Description

Configuration structure for controlling statistical evaluation of film analysis.
This structure allows customization of how the statistical evaluator processes and outputs data derived from frame-by-frame shot classification. It includes control parameters for sampling, entropy analysis, cut detection, and export settings.
The documentation for this struct was generated from the following file:

- include/UserStructs.hpp

5.5 HaarDetector Class Reference

Detects visual features (e.g., faces) in an image using Haar cascade models.
`#include <HaarDetector.hpp>`

Public Member Functions

- HaarDetector (std::string &modelPath)

  Constructs the detector and immediately loads the model.
- **HaarDetector** ()

  Constructs the detector without loading a model initially.
- ∼**HaarDetector** ()=default

  destructor.
- void loadModel (const std::string &modelPath)

  Loads a Haar cascade model from the specified file path.
- std::vector< cv::Rect > detect (const cv::Mat &image)

  Detects features in the given image.

Private Attributes

- cv::CascadeClassifier **cascade**

  The loaded Haar cascade classifier used for detection.

5.5.1 Detailed Description

Detects visual features (e.g., faces) in an image using Haar cascade models.
This class provides a simple interface for loading a Haar cascade model and applying it to an input image to detect features such as faces, bodies, etc. The detector returns a list of `DetectedFeature` results, each containing a label and bounding box.
It may be beneficial to sort the output vector by bounding box size (e.g., largest first), to simplify downstream feature selection or analysis.

See also

HaarFeature

5.5.2 Constructor & Destructor Documentation

5.5.2.1 HaarDetector()

```
HaarDetector::HaarDetector (
            std::string & modelPath)
```

Constructs the detector and immediately loads the model.

Parameters

modelPath	Path to the Haar cascade XML model file.

5.5.3 Member Function Documentation

5.5.3.1 detect()

```
std::vector< cv::Rect > HaarDetector::detect (
            const cv::Mat & image)
```

Detects features in the given image.
Applies the currently loaded Haar cascade model to the image and returns a list of bounding boxes with associated labels.

Parameters

image	The image in which to detect features.

Returns

A vector of `std::vector<cv::Rect>` representing all detected faces.

5.5.3.2 loadModel()

```
void HaarDetector::loadModel (
            const std::string & modelPath)
```

Loads a Haar cascade model from the specified file path.

Parameters

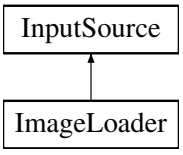
modelPath	Path to the Haar cascade XML model file.

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

- include/HaarDetector.hpp
- src/HaarDetector.cpp

5.6 ImageLoader Class Reference

Concrete implementation of InputSource for loading a single image.
`#include <FileLoader.hpp>`
Inheritance diagram for ImageLoader:



Public Member Functions

- **ImageLoader** (const std::string &path)
- bool hasNextFrame () const override

  Checks if the image is still available to be returned.
- cv::Mat & nextFrame () override

  Returns the loaded image.
- double getCurrentTimestamp () const override

  Returns the timestamp of the image (usually 0).
- InputSource (const std::string &path)

  Constructs an input source with a file path.

Public Member Functions inherited from InputSource

- InputSource (const std::string &path)

  Constructs an input source with a file path.
- virtual ~**InputSource** ()=default

  Virtual destructor for polymorphic use.

Private Member Functions

- void **loadImagePathsFromDirectory** (const std::string &directory)

Private Attributes

- cv::Mat **image**
- bool **loaded** = false
- bool **frameReturned** = false
- std::vector< std::string > **image_paths**
- int **current_frame_index** = 0

Additional Inherited Members

Protected Attributes inherited from InputSource

- std::string **source_path**

  Path to the input source file (image or video)

5.6.1 Detailed Description

Concrete implementation of InputSource for loading a single image.
Provides a way to treat a single image file as a frame source. Once the image is returned, subsequent calls indicate no more frames are available.

5.6.2 Member Function Documentation

5.6.2.1 getCurrentTimestamp()

```
double ImageLoader::getCurrentTimestamp () const  [override], [virtual]
```

Returns the timestamp of the image (usually 0).

Returns

Fixed timestamp value (e.g., 0 ms).

Implements InputSource.

5.6.2.2 hasNextFrame()

```
bool ImageLoader::hasNextFrame () const  [override], [virtual]
```

Checks if the image is still available to be returned.

Returns

True if the image hasn't been returned yet.

Implements InputSource.

5.6.2.3 InputSource()

```
InputSource::InputSource (
            const std::string & path)  [inline], [explicit]
```

Constructs an input source with a file path.

Parameters

path	Path to the image or video file.

5.6.2.4 nextFrame()

```
cv::Mat & ImageLoader::nextFrame ()  [override], [virtual]
```

Returns the loaded image.

Returns

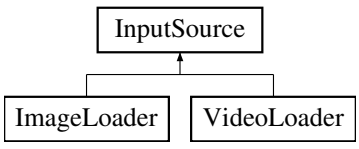
Reference to the loaded image (`cv::Mat`).

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

- include/FileLoader.hpp
- src/FileLoader.cpp

5.7 InputSource Class Reference

Abstract base class for loading image or video input in a unified way.
`#include <FileLoader.hpp>`
Inheritance diagram for InputSource:



Public Member Functions

- InputSource (const std::string &path)

  Constructs an input source with a file path.
- virtual ~**InputSource** ()=default

  Virtual destructor for polymorphic use.
- virtual bool hasNextFrame () const =0

  Checks whether there is a next frame to process.
- virtual cv::Mat & nextFrame ()=0

  Returns the next frame.
- virtual double getCurrentTimestamp () const =0

  Returns the current timestamp of the frame (in ms).

Protected Attributes

- std::string **source_path**

  Path to the input source file (image or video)

5.7.1 Detailed Description

Abstract base class for loading image or video input in a unified way.
This class defines a common interface for different input sources (image or video), allowing the rest of the application to interact with frames in a uniform manner. It supports checking for availability of the next frame, retrieving the frame, and (optionally) querying its timestamp.
Designed to make it easy to switch between static image input and sequential video input.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 InputSource()

```
InputSource::InputSource (
            const std::string & path)  [inline], [explicit]
```

Constructs an input source with a file path.

Parameters

path	Path to the image or video file.

5.7.3 Member Function Documentation

5.7.3.1 getCurrentTimestamp()

```
virtual double InputSource::getCurrentTimestamp () const  [pure virtual]
```

Returns the current timestamp of the frame (in ms).

Returns

Timestamp in milliseconds.

Implemented in ImageLoader, and VideoLoader.

5.7.3.2 hasNextFrame()

```
virtual bool InputSource::hasNextFrame () const  [pure virtual]
```

Checks whether there is a next frame to process.

Returns

True if a frame is available, false otherwise.

Implemented in ImageLoader, and VideoLoader.

5.7.3.3 nextFrame()

```
virtual cv::Mat & InputSource::nextFrame ()  [pure virtual]
```

Returns the next frame.

Returns

A reference to the next `cv::Mat` frame.

Implemented in ImageLoader, and VideoLoader.
The documentation for this class was generated from the following file:

- include/FileLoader.hpp

5.8 openCV_lib Class Reference

Public Member Functions

- void **HelloWorld** (const char *)

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

- openCV_lib/openCV_lib.hpp
- openCV_lib/openCV_lib.cpp

5.9 openCV_libPriv Class Reference

Public Member Functions

- void **HelloWorldPriv** (const char *)

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

- openCV_lib/openCV_libPriv.hpp
- openCV_lib/openCV_lib.cpp

5.10 Preprocessing Class Reference

Class responsible for applying preprocessing operations to a frame.
`#include <FileLoader.hpp>`

Public Member Functions

- Preprocessing (const cv::Mat frame)

  Constructs the preprocessing unit with an input frame.
- **Preprocessing** ()

  Default constructor.
- ~**Preprocessing** ()

  Destructor.
- void LoadFrame (cv::Mat &image)

  Loads a new frame into the processor.
- void resizeImage (int const rows, int const cols)

  Resizes the internal image to the specified dimensions.
- void **toGrayscale** ()

  Converts the internal image to grayscale.
- void **equalizeHistogram** ()

  Applies histogram equalization to enhance image contrast.
- cv::Mat & GetProcessedImage ()

  Returns the processed image.

Private Attributes

- cv::Mat **image**

  Internal image being processed.

5.10.1 Detailed Description

Class responsible for applying preprocessing operations to a frame.
This class loads a frame and provides an interface to process it (e.g., grayscale, histogram equalization, resizing, denoising, etc.).
Intended to be extended with actual image preprocessing methods for analysis or detection.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Preprocessing()

```
Preprocessing::Preprocessing (
            const cv::Mat frame)  [inline], [explicit]
```

Constructs the preprocessing unit with an input frame.

Parameters

frame	Frame to preprocess.

5.10.3 Member Function Documentation

5.10.3.1 GetProcessedImage()

```
cv::Mat & Preprocessing::GetProcessedImage ()
```

Returns the processed image.

Returns

Reference to the processed image.

5.10.3.2 LoadFrame()

```
void Preprocessing::LoadFrame (
            cv::Mat & image)
```

Loads a new frame into the processor.

Parameters

image	Frame to load for processing.

5.10.3.3 resizeImage()

```
void Preprocessing::resizeImage (
            int const rows,
            int const cols)
```

Resizes the internal image to the specified dimensions.

Parameters

rows	Desired number of rows (height).
cols	Desired number of columns (width).

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

- include/FileLoader.hpp
- src/FileLoader.cpp

5.11 ResultDisplayer Class Reference

Provides graphical or textual output for film analysis results.

```
#include <ResultDisplayer.hpp>
```

Public Member Functions

- ResultDisplayer (const FilmStatistics &stats)

  Constructs the displayer with a copy of the film statistics.
- ~**ResultDisplayer** ()=default

  Default destructor.
- cv::Mat **GetPlot** (std::vector< std::pair< double, std::map< ShotType, double > > > &data_series, std::string xlabel, std::string ylabel, std::string title)
- cv::Mat **GetPlot** (std::vector< std::pair< double, ShotType > >, std::string xlabel, std::string ylabel, std::string title)
- cv::Mat **GetPlot** (std::vector< std::pair< double, double > > &data_series, std::string xlabel, std::string ylabel, std::string title)
- cv::Mat **GetPlot** (std::vector< std::pair< double, bool > > &data_series, std::string xlabel, std::string ylabel, std::string title)

Private Attributes

- FilmStatistics **film_stats**

  Collected statistics about the film (copied on construction)

5.11.1 Detailed Description

Provides graphical or textual output for film analysis results.
The ResultDisplayer is responsible for visualizing the data collected in the FilmStatistics object. This includes summaries, distributions of shot types, timelines, or other outputs meant for the user.
This class currently stores a copy of the FilmStatistics object. For performance or design flexibility, a reference-based approach may also be considered depending on use case.
Future implementations may include rendering to GUI, charts, or image overlays.

See also

FilmStatistics

5.11.2 Constructor & Destructor Documentation

5.11.2.1 ResultDisplayer()

```
ResultDisplayer::ResultDisplayer (
            const FilmStatistics & stats)  [inline]
```

Constructs the displayer with a copy of the film statistics.

Parameters

stats	The FilmStatistics object containing the processed analysis data.

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

- include/ResultDisplayer.hpp
- src/ResultDisplayer.cpp

5.12 ShotClassifier Class Reference

Public Member Functions

- **ShotClassifier** (int smallest_thresh=5000, int closeup_thresh=30000)
- int **classify** (const std::vector< cv::Rect > &shot_features)

Private Attributes

- int **smallest_face_threshold**
- int **medium_face_threshold**
- int **closeup_face_threshold**

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

- include/ShotClassifier.hpp
- src/ShotClassifier.cpp

5.13 ShotEvaluator Class Reference

Public Member Functions

- **ShotEvaluator** (HaarDetector &frontal, HaarDetector &profile, HaarDetector &eye, ShotClassifier &face↩ Classifier, int eyeThreshold)
- ClassificationResult **evaluate** (const cv::Mat &image, std::vector< cv::Rect > &allFaces, std::vector< cv::↩ Rect > &eyes)

Private Attributes

- HaarDetector & **frontal_face_detector**
- HaarDetector & **profile_face_detector**
- HaarDetector & **eye_detector**
- ShotClassifier & **face_classifier**
- ShotClassifier **eye_classifier**

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

- include/ShotEvaluator.hpp
- src/ShotEvaluator.cpp

5.14 ShotFeatures Struct Reference

Represents extracted geometric and area-based properties from a video frame.
`#include <UserStructs.hpp>`

Public Attributes

- int **object_count** = 0

  Number of detected objects in the frame.
- double **largest_object_area** = 0.

  Area of the largest object detected.
- double **total_object_area** = 0.

  Sum of all object areas.
- double **total_area** = 0.

  Total area of the frame (width ∗ height)
- std::vector< cv::Point2f > **object_centers**

  Center points of detected objects, ordered by size.

5.14.1 Detailed Description

Represents extracted geometric and area-based properties from a video frame.
ShotFeatures holds intermediate data used for shot type classification. It includes statistics like the number of detected objects, the total and largest object area, and the coordinates of their centers.
The `object_centers` vector is ordered by object size, with the largest first.
The documentation for this struct was generated from the following file:

- include/UserStructs.hpp

5.15 TestDatasetEval Class Reference

Evaluates classification accuracy against a labeled test dataset.
`#include <TestDatasetEval.hpp>`

Public Member Functions

- **TestDatasetEval** (ShotType type)
- bool check (ShotType input_type)

  Adds a predicted image classification result to the evaluation set.
- double GetEvalResult ()

  Computes and returns the evaluation result (e.g., accuracy).

Private Attributes

- ShotType **desired_type**
- int **true_detect**
- int **counter**

5.15.1 Detailed Description

Evaluates classification accuracy against a labeled test dataset.
The `TestDatasetEval` class is designed to compare predicted classification results with a predefined set of ground truth labels. It supports loading the ground truth data, adding prediction samples, and computing an evaluation score (e.g., accuracy).
This is useful for testing and validating the performance of the shot classifier on labeled datasets.
Example usage:

```
TestDatasetEval evaluator;
evaluator.loadGroundTruth("ground_truth.csv");
evaluator.addImageSample(predictedLabel);
double accuracy = evaluator.GetEvalResult();
```

Author

Marek Tatýrek

Date

2025

See also

ClassificationResult

5.15.2 Member Function Documentation

5.15.2.1 check()

```
bool TestDatasetEval::check (
            ShotType input_type)  [inline]
```

Adds a predicted image classification result to the evaluation set.

Parameters

Image	The image to be evaluated (prediction logic assumed to be applied externally).

5.15.2.2 GetEvalResult()

```
double TestDatasetEval::GetEvalResult ()  [inline]
```

Computes and returns the evaluation result (e.g., accuracy).

Returns

A floating-point score representing classifier performance.

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

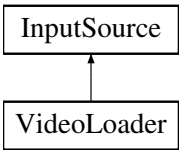
- include/TestDatasetEval.hpp

5.16 VideoLoader Class Reference

Placeholder for a video-based implementation of InputSource.

`#include <FileLoader.hpp>`

Inheritance diagram for VideoLoader:



Public Member Functions

- **VideoLoader** (const std::string &path)
- bool hasNextFrame () const override

  Checks if the image is still available to be returned.
- cv::Mat & nextFrame () override

  Returns the loaded image.
- double getCurrentTimestamp () const override

  Returns the timestamp of the image (usually 0).
- InputSource (const std::string &path)

  Constructs an input source with a file path.

Public Member Functions inherited from InputSource

- InputSource (const std::string &path)

  Constructs an input source with a file path.
- virtual ~**InputSource** ()=default

  Virtual destructor for polymorphic use.

Private Member Functions

- void **openVideoFromPath** (const std::string &path)

Private Attributes

- std::string **video_path**
- cv::VideoCapture **video**
- cv::Mat **current_frame**
- int **current_frame_index**

Additional Inherited Members

Protected Attributes inherited from InputSource

- std::string **source_path**

  Path to the input source file (image or video)

5.16.1 Detailed Description

Placeholder for a video-based implementation of InputSource.
This class will provide functionality to iterate over frames from a video file. Not implemented yet.

5.16.2 Member Function Documentation

5.16.2.1 getCurrentTimestamp()

```
double VideoLoader::getCurrentTimestamp () const  [override], [virtual]
```

Returns the timestamp of the image (usually 0).

Returns

Fixed timestamp value (e.g., 0 ms).

Implements InputSource.

5.16.2.2 hasNextFrame()

```
bool VideoLoader::hasNextFrame () const  [override], [virtual]
```

Checks if the image is still available to be returned.

Returns

True if the image hasn't been returned yet.

Implements InputSource.

5.16.2.3 InputSource()

```
InputSource::InputSource (
            const std::string & path)  [inline], [explicit]
```

Constructs an input source with a file path.

Parameters

path	Path to the image or video file.

5.16.2.4 nextFrame()

```
cv::Mat & VideoLoader::nextFrame ()  [override], [virtual]
```

Returns the loaded image.

Returns

Reference to the loaded image (`cv::Mat`).

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

- include/FileLoader.hpp
- src/FileLoader.cpp

Chapter 6

File Documentation

6.1 FileLoader.hpp

```
00001 //
00002 //  FileLoader.hpp
00003 //  Film_type_classifier
00004 //
00005 //  Created by Peter ... on 20.05.2025.
00006 //
00007
00008 #ifndef FileLoader_hpp
00009 #define FileLoader_hpp
00010
00011 #include <stdio.h>
00012 #include <opencv2/opencv.hpp>
00013 #include <iostream>
00014 #include <opencv2/highgui.hpp>
00015 #include <opencv2/imgproc.hpp>
00016 #include <string>
00017 #include <vector>
00018
00030 class InputSource
00031 {
00032 protected:
00033     std::string source_path;
00034
00035 public:
00036
00037
00042     explicit InputSource(const std::string& path) : source_path(path) {}
00043
00047     virtual ~InputSource() = default;
00048
00053     virtual bool hasNextFrame() const = 0;
00054
00059     virtual cv::Mat& nextFrame() = 0;
00060
00065     virtual double getCurrentTimestamp() const = 0;
00066 };
00067
00068
00069
00070
00071
00079 class ImageLoader : public InputSource
00080 {
00081 private:
00082     cv::Mat image; // Loaded image
00083     bool loaded = false;
00084     bool frameReturned = false;
00085
00086     std::vector<std::string> image_paths;
00087     int current_frame_index = 0;    // reset indexu
00088
00089
00090     void loadImagePathsFromDirectory(const std::string& directory);
00091
00092 public:
00093
00094     using InputSource::InputSource;
00095
00096     ImageLoader(const std::string& path) : InputSource(path)
00097     {
00098         loadImagePathsFromDirectory(path);
```

```
00099     }
00100
00105     bool hasNextFrame() const override;
00106
00111     cv::Mat& nextFrame() override;
00112
00117     double getCurrentTimestamp() const override;
00118 };
00119
00127 class VideoLoader : public InputSource
00128 {
00129     std::string video_path;
00130     cv::VideoCapture video;
00131
00132     cv::Mat current_frame;
00133
00134     int current_frame_index;
00135
00136     void openVideoFromPath(const std::string & path);
00137
00138 public:
00139     using InputSource::InputSource;
00140
00141     VideoLoader(const std::string& path) : InputSource(path)
00142     {
00143         openVideoFromPath(path);
00144     }
00145
00150     bool hasNextFrame() const override;
00151
00156     cv::Mat& nextFrame() override;
00157
00162     double getCurrentTimestamp() const override;
00163
00164
00165 };
00166
00167
00168
00169
00170
00171
00172
00173
00174
00184 class Preprocessing
00185 {
00186     cv::Mat image;
00187
00188 public:
00193     explicit Preprocessing(const cv::Mat frame) : image(frame) {}
00194
00198     explicit Preprocessing() {}
00199
00203     ~Preprocessing();
00204
00209     void LoadFrame(cv::Mat& image);
00210
00216     void resizeImage(int const rows, int const cols);
00220     void toGrayscale();
00224     void equalizeHistogram();
00225
00230     cv::Mat& GetProcessedImage();
00231
00232     // Future: add methods for specific preprocessing steps (blur, resize, etc.)
00233 };
00234
00235 #endif /* FileLoader_hpp */
```

6.2 FilmStatisticEval.hpp

```
00001 //
00002 //  FilmStatisticEval.hpp
00003 //  Film_type_classifier
00004 //
00005 //  Created by Marek Tatýrek on 21.05.2025.
00006 //
00007
00008 #ifndef FilmStatisticEval_hpp
00009 #define FilmStatisticEval_hpp
00010
00011 #include <stdio.h>
00012 #include <opencv2/opencv.hpp>
00013 #include <numeric>
```

```
00014 #include <fstream>
00015 #include "UserStructs.hpp"
00016
00048 class FilmStatistics
00049 {
00053     std::map<ShotType, int> shot_counts;
00054
00059     std::vector<std::pair<double, std::map<ShotType, double>> prob_timeline;
00060
00065     std::vector<std::pair<double, ShotType> shot_type_timeline;
00066
00071     std::vector<std::pair<double, double> enthropy_timeline;
00072
00077     std::vector<std::pair<double, double> enthropy_variance_timeline;
00078
00083     std::vector<std::pair<double, bool> cut_detection_timeline;
00084
00088     std::deque<ClassificationResult> oversampling_sliding_window;
00089
00093     std::deque<ClassificationResult> entropy_sliding_window;
00094
00098     std::deque<double> entropy_variance_sliding_window;
00099
00103     int total_frames = 0;
00104
00108     int evaluated_frames = 0;
00109
00113     ClassificationResult current_sample_oversampled;
00114
00118     double entropy = 0.0;
00119
00123     double timestamp_ms = 0.0;
00124
00128     double entropy_variance = 0.0;
00129
00133     FilmStatisticsEvalConfig config;
00134
00138     int start_delay = 0;
00139
00143     double frame_time_measurement;
00144
00149     void normalizeProbs(std::map<ShotType, double> & aggregated_probs);
00150
00156     void aggregateSlidingWindowProbs(std::deque<ClassificationResult> & sliding_window,
      std::map<ShotType, double> & aggregated_probs);
00157
00163     void findShotTypeMaxProb(std::map<ShotType, double> & aggregated_probs, ClassificationResult &
      sample);
00164
00168     void oversampleInputData();
00169
00173     void computeEntropy();
00174
00178     void computeEntropyVariance();
00179
00183     void outputStatistics();
00184
00189     void appendSampleToTimeline(std::vector<std::pair<double, std::map<ShotType, double>> & timeline);
00190
00195     void appendSampleToTimeline(std::vector<std::pair<double, ShotType> & timeline);
00196
00201     void appendEntropyToTimeline(std::vector<std::pair<double, double> & timeline);
00202
00207     void appendEntropyVarianceToTimeline(std::vector<std::pair<double, double> & timeline);
00208
00213     void appendCutDetectionToTimeline(std::vector<std::pair<double, bool> & timeline);
00214
00215 public:
00219     FilmStatistics() = default;
00220
00225     void addConfigurationStruct(FilmStatisticsEvalConfig const & cfg);
00226
00233     void addFrameResult(const double timestampMs, const ClassificationResult & result);
00234
00242     void exportToCSV(const std::string& path) const;
00243
00247     void printSummary() const;
00248
00249
00250     // -- temporary --
00251 //    /**
00252 //     * @brief Gets the entropy timeline.
00253 //     * @return Reference to vector of timestamp and entropy pairs.
00254 //     */
00255 //    std::vector<std::pair<double, double> & getEntropy(){return enthropy_timeline; };
00256 //
00257 //    /**
```

```
00258 //     * @brief Gets the entropy variance timeline.
00259 //     * @return Reference to vector of timestamp and entropy variance pairs.
00260 //     */
00261 //    std::vector<std::pair<double, double» & getEntropyVAriance(){return enthropy_variance_timeline;
      };
00262 //
00263 //    /**
00264 //     * @brief Gets the shot type timeline.
00265 //     * @return Reference to vector of timestamp and shot type pairs.
00266 //     */
00267 //    std::vector<std::pair<double, ShotType» & getShotType(){return shot_type_timeline;}
00268 //
00269 //    std::vector<std::pair<double, std::map<ShotType, double»> & getProbTimeline(){return
      prob_timeline;};
00270 //
00271     // -- temporary --
00272 };
00273
00274 #endif /* FilmStatisticEval_hpp */
```

6.3 HaarDetector.hpp

```
00001 //
00002 //  HaarDetector.hpp
00003 //  Film_type_classifier
00004 //
00005 //  Created by Miroslaw on 20.05.2025.
00006 //
00007
00008 #ifndef HaarDetector_hpp
00009 #define HaarDetector_hpp
00010
00011 #include <stdio.h>
00012 #include <opencv2/opencv.hpp>
00013
00027 class HaarDetector
00028 {
00029     cv::CascadeClassifier cascade;
00030
00031 public:
00036     HaarDetector(std::string& modelPath);
00037
00038
00042     HaarDetector() {}
00043
00047     ~HaarDetector() = default;
00048
00053     void loadModel(const std::string& modelPath);
00054
00064     std::vector<cv::Rect> detect(const cv::Mat& image); // maybe it will be fine to sort the vector
      from biggest BB, so we gonna have easier job afterwards
00065 };
00066
00067 #endif /* HaarDetector_hpp */
```

6.4 ResultDisplayer.hpp

```
00001 //
00002 //  ResultDisplayer.hpp
00003 //  Film_type_classifier
00004 //
00005 //  Created by Peter... on 21.05.2025.
00006 //
00007
00008 #ifndef ResultDisplayer_hpp
00009 #define ResultDisplayer_hpp
00010
00011 #include <stdio.h>
00012 #include "FilmStatisticEval.hpp"
00013
00030 class ResultDisplayer
00031 {
00032     FilmStatistics film_stats;
00033
00034 public:
00039     ResultDisplayer(const FilmStatistics& stats) : film_stats(stats) {}
00040
00044     ~ResultDisplayer() = default;
00045
00046     cv::Mat GetPlot(std::vector<std::pair<double, std::map<ShotType, double»> & data_series,
      std::string xlabel, std::string ylabel, std::string title);
```

```
00047     cv::Mat GetPlot(std::vector<std::pair<double, ShotType», std::string xlabel, std::string ylabel,
      std::string title);
00048     cv::Mat GetPlot(std::vector<std::pair<double, double» & data_series, std::string xlabel,
      std::string ylabel, std::string title);
00049     cv::Mat GetPlot(std::vector<std::pair<double, bool» & data_series, std::string xlabel, std::string
      ylabel, std::string title);
00050 };
00051
00052 #endif /* ResultDisplayer_hpp */
```

6.5 SceneChangeDetect.hpp

```
00001 //
00002 //  sceneChangeDetector.hpp
00003 //  Film_type_classifier
00004 //
00005 //  Created by Marek Tatýrek on 14.07.2025.
00006 //
00007
00008 #ifndef sceneChangeDetector_hpp
00009 #define sceneChangeDetector_hpp
00010
00011 #include <stdio.h>
00012 #include <opencv2/opencv.hpp>
00013
00014 bool isSceneChanged(const cv::Mat& prev, const cv::Mat& curr, double threshold = 0.5, int
      pixelDiffThreshold = 60);
00015
00016 #endif /* sceneChangeDetector_hpp */
```

6.6 ShotClassifier.hpp

```
00001 #ifndef ShotClassifier_hpp
00002 #define ShotClassifier_hpp
00003
00004 #include <vector>
00005 #include <opencv2/opencv.hpp>
00006
00007 class ShotClassifier
00008 {
00009     int smallest_face_threshold; // minimal area for smallest face
00010     int medium_face_threshold;   // minimal area for medium shot
00011     int closeup_face_threshold;  // minimal area for closeup
00012
00013 public:
00014     ShotClassifier(int smallest_thresh = 5000, int closeup_thresh = 30000);
00015
00016     int classify(const std::vector<cv::Rect>& shot_features);
00017 };
00018
00019 #endif /* ShotClassifier_hpp */
```

6.7 ShotEvaluator.hpp

```
00001 #include <opencv2/opencv.hpp>
00002 #include "HaarDetector.hpp"
00003 #include "ShotClassifier.hpp"
00004
00005 // Assuming ShotType and ClassificationResult are defined in a separate header
00006 #include "UserStructs.hpp"
00007
00008 class ShotEvaluator {
00009 public:
00010     ShotEvaluator(HaarDetector& frontal, HaarDetector& profile, HaarDetector& eye, ShotClassifier&
      faceClassifier, int eyeThreshold);
00011
00012     ClassificationResult evaluate(const cv::Mat& image, std::vector<cv::Rect>& allFaces,
      std::vector<cv::Rect>& eyes);
00013
00014 private:
00015     HaarDetector& frontal_face_detector;
00016     HaarDetector& profile_face_detector;
00017     HaarDetector& eye_detector;
00018     ShotClassifier& face_classifier;
00019     ShotClassifier eye_classifier;
00020 };
```

6.8 TestDatasetEval.hpp

```
00001 //
00002 //  TestDatasetEval.hpp
00003 //  Film_type_classifier
00004 //
00005 //  Created by Marek Tatýrek on 06.06.2025.
00006 //
00007
00008 #ifndef TestDatasetEval_hpp
00009 #define TestDatasetEval_hpp
00010
00011 #include <stdio.h>
00012 #include <opencv2/opencv.hpp>
00013 #include <iostream>
00014 #include "UserStructs.hpp"
00015
00016 #endif /* TestDatasetEval_hpp */
00017
00018
00042 class TestDatasetEval
00043 {
00044
00045     ShotType desired_type;
00046
00047     int true_detect;
00048     int counter;
00049
00050 public:
00051
00052     TestDatasetEval(ShotType type) : desired_type(type)
00053     {
00054         true_detect = 0;
00055         counter = 0;
00056     }
00057
00062     bool check(ShotType input_type)
00063     {
00064         counter ++;
00065         if(input_type == desired_type)
00066         {
00067             true_detect ++;
00068             return true;
00069         }
00070         return false;
00071     }
00072
00077     double GetEvalResult()
00078     {
00079         return static_cast<double>(true_detect)/static_cast<double>(counter);
00080     }
00081 };
```

6.9 UserStructs.hpp

```
00001 //
00002 //  UserStructs.hpp
00003 //  Film_type_classifier
00004 //
00005 //  Created by Marek Tatýrek on 06.06.2025.
00006 //
00007
00008 #ifndef UserStructs_hpp
00009 #define UserStructs_hpp
00010
00011 #include <stdio.h>
00012 #include <opencv2/opencv.hpp>
00013 #include <iostream>
00014
00022 struct DetectedFeature {
00023     std::string label;
00024     cv::Rect boundingBox;
00025 };
00026
00037 struct ShotFeatures {
00038     int object_count = 0;
00039
00040     double largest_object_area = 0.;
00041     double total_object_area = 0.;
00042     double total_area = 0.;
00043
00044     // for good shot classification + we can add some statistical metrics
00045     std::vector<cv::Point2f> object_centers;
```

```
00046 };
00047
00054 enum class ShotType {
00055     CLOSE_UP,
00056     MEDIUM,
00057     WIDE,
00058     UNKNOWN
00059 };
00060
00070 struct ClassificationResult {
00071     ShotType predictedType = ShotType::UNKNOWN;
00072     std::map<ShotType, double> probabilities;
00073 };
00074
00083 struct FilmStatisticsEvalConfig {
00084     size_t input_step = 1;
00085     size_t input_oversample = 1;
00086
00087     size_t entropy_window_size = 20;
00088     size_t entropy_variance_window_size = 30;
00089
00090     double cut_detect_entropy_threshold = 1;
00091
00092     bool output_ratios_series = false;
00093     size_t output_shot_type_time_series;
00094     size_t output_prob_time_series = 1;
00095     size_t output_entropy_time_series = 1;
00096     size_t output_entropy_variance_time_series = 1;
00097     size_t output_cut_detection_time_series = 1;
00098
00099     double cut_detection_entropy_treshold = 1.3;
00100     size_t cut_detection_history_window_size = 15;
00101     double cut_detection_entropy_diff_treshold = 0.3;
00102 };
00103 #endif /* UserStructs_hpp */
00104
```

6.10 openCV_lib.hpp

```
00001 //
00002 //  openCV_lib.hpp
00003 //  openCV_lib
00004 //
00005 //  Created by Marek Tatýrek on 11.03.2025.
00006 //
00007
00008 #ifndef openCV_lib_
00009 #define openCV_lib_
00010
00011 /* The classes below are exported */
00012 #pragma GCC visibility push(default)
00013
00014 class openCV_lib
00015 {
00016     public:
00017     void HelloWorld(const char *);
00018 };
00019
00020 #pragma GCC visibility pop
00021 #endif
```

6.11 openCV_libPriv.hpp

```
00001 //
00002 //  openCV_libPriv.hpp
00003 //  openCV_lib
00004 //
00005 //  Created by Marek Tatýrek on 11.03.2025.
00006 //
00007
00008 /* The classes below are not exported */
00009 #pragma GCC visibility push(hidden)
00010
00011 class openCV_libPriv
00012 {
00013     public:
00014     void HelloWorldPriv(const char *);
00015 };
00016
00017 #pragma GCC visibility pop
```