AUFTRAGGEBER:

〉 FRAUNHOFER IOSB

PSE
SS
2021

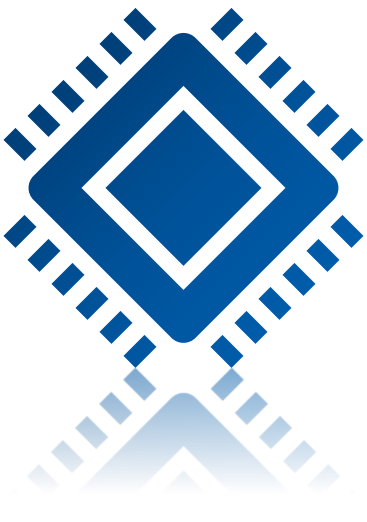
BETREUER:

〉 THOMAS POLLOK & STEFAN WOLF

Rapid Classification AI Trainer

ENTWURFSHEFT

〉 ADRIAN NELSON
〉 ANDREAS OTT
〉 DARIA BIDENKO
〉 JONAS WILLE
〉 PAUL SCHAARSCHMIDT
〉 SASCHA ROLINGER



Inhaltsverzeichnis

1 Architekturbeschreibung

Die Software *Rapid Classification AI Trainer* (**RCAIT**) besteht aus 3 Hauptkomponenten. Einer graphischen Benutzeroberfläche, dem Modell, das die zugrundeliegenden Datenstrukturen umfasst, und dem Controller der als Vermittler zwischen Oberfläche und Modell wirkt. Das Modell ist für das Laden und Verwalten von Projekten sowie Klassifikationsmodellen zuständig, des Weiteren führt es auch Berechnungen aus und stellt die Ergebnisse dieser zur weiteren Verwendung bereit. Klassen des Modells sind im Paket Model zusammengefasst. Die Fenster und Dialoge der graphischen Benutzeroberfläche dienen der sinnvollen Darstellung von Modelldaten (z.B. Konfusionsmatrix, Verlustkurve), und nehmen auch Benutzereingaben entgegen, um diese dann an den Controller weiterzuleiten. Klassen der grafischen Benutzeroberfläche sind im Paket View zusammengefasst.

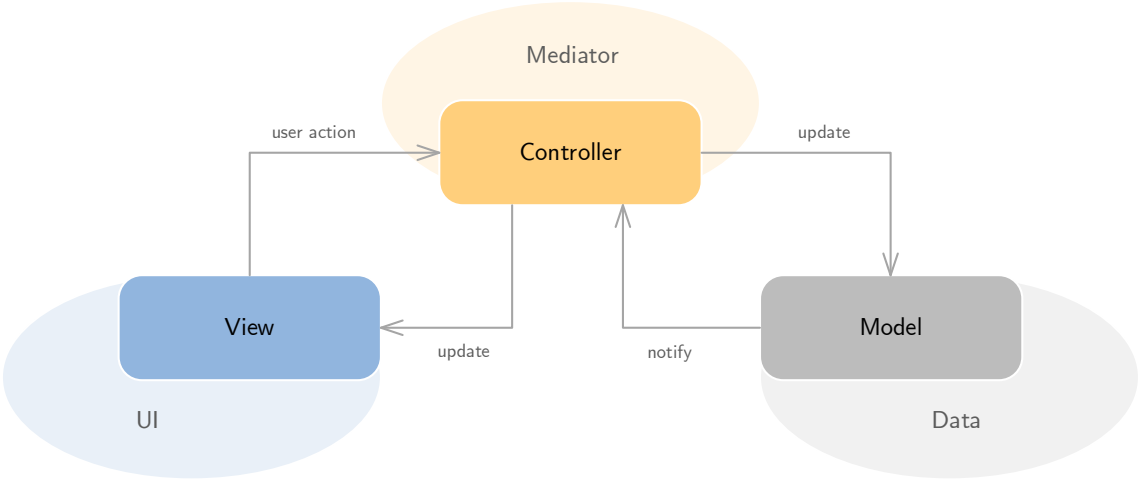


Abbildung 1: Genereller Aufbau des Model View Controllers

Um eine hohe Kopplung zwischen dem Modell und der View zu vermeiden, wird ein sogenannter Controller dazwischengeschaltet. Dieser empfängt die Benutzereingaben, interpretiert sie im Kontext des Modells, und fordert anschließend das Modell auf diese umzusetzen. Der Controller übernimmt auch die Steuerung für Aspekte der Benutzeroberfläche, die nicht direkt mit Modelldaten zusammenhängen, wie das Öffnen und Schließen von Fenstern bzw. Anzeigen. Klassen des Controllers sind im Paket Controller zusammengefasst.

Das Beobachter Muster prägt sich stark in der Benutzeroberfläche – Controller und Controller – Modell Kommunikation aus und ist auch ein Wichtiger Bestandteil der hier verwendeten MVC Architektur. In Qt wird eine Subjekt Beobachter Relation mittels Signals und Slots realisiert.

In dieser Abbildung sind die wichtigsten Bestandteile der Software aufgefasst. Die Klassen, die als Zugangspunkt für ihr jeweiliges Paket zu Verfügung stehen, werden durch Kasten repräsentiert. Andere paketzugehörige Klassen sind in Textform aufgelistet. Zwischen beiden Klassenarten wird in den folgenden Kapiteln nicht explizit unterschieden, die Darstellungen hier dient lediglich der Übersichtlichkeit.

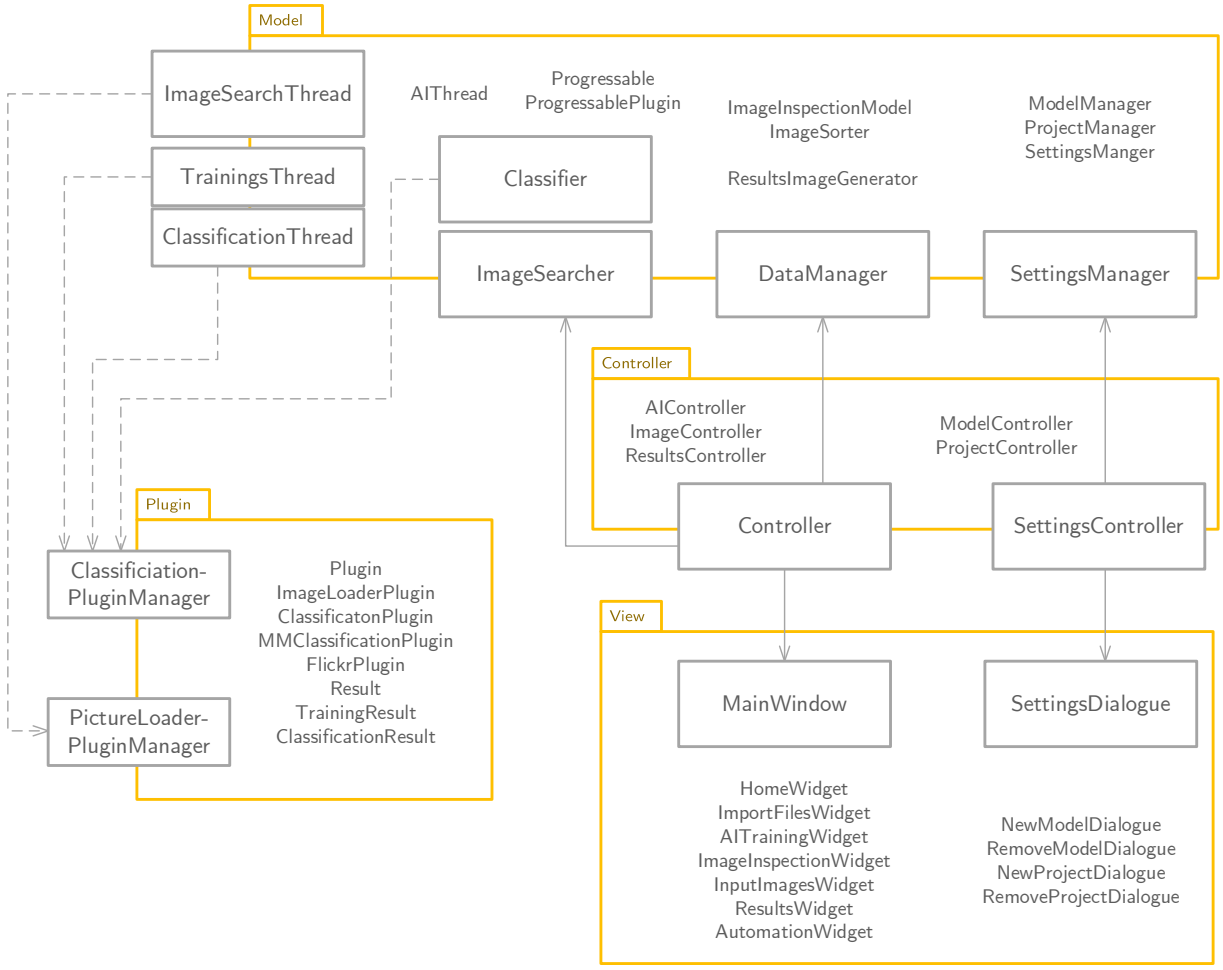


Abbildung 2 : Softwarearchitektur

Innerhalb der Beschreibung der einzelnen „Model“, „View“, „Controller“ und „Plugin“ Pakete, werden Relationen und Abhängigkeiten zwischen den einzelnen Klassen genauer dargestellt und beschrieben. Der Hauptfokus ist hier die Repräsentation der MVC Architektur und deren Einbindung mit den anderen Softwarekomponenten. Die entkoppelnde Funktionsweise der MVC Architektur lässt sich an den Abhängigkeiten erkennen. Eine wichtige Anforderung aus dem Pflichtenheft ist die Erweiterbarkeit der Software durch neue Klassifikatoren und Datenquellen. Entsprechend gibt es eine Plugin Architektur, die das Hinzufügen weiterer Funktionalität – auch nach der Auslieferung der Software – unterstützt.

Das `ClassificationPlugin` sowie das `PictureLoaderPlugin` Interface erweitern das Allgemeine `Plugin` Interface und dienen dabei als Schnittstelle für die zwei verschiedene Arten von Plugins. Weitere Trainingsmethoden oder Datenquellen implementieren dann diese Interfaces entsprechend. Wie im Pflichtenheft beschrieben werden Plugin-Implementierungen für MMClassification und Flickr bereitgestellt, diese befinden sich in den `MMClassificationPlugin` und `Flickrplugin` Klassen. Die Verwaltung und Verwendung aller Plugins wird über den `PluginManager` verlaufen.

Anmerkung: Aus Gründen der Übersichtlichkeit wird die Vererbung von `QObject` auf alle hier verwendeten Qt Klassen weggelassen.
Für Signale und Slots definieren wir für dieses Heft folgende Konvention der Schreibweise:

```
Signal: > sig_[SignalName]([Parameter])
Slot: $ slot_[SlotName]([Parameter])
```

Zudem werden Signale und dazu zugehörige Slots nach unserer selbstauferlegten Konvention gleich (oder zumindest ähnlich) bezeichnet, um die Lesbarkeit der Klassenstrukturen zu erhöhen.

2 Model

Das *Model* hat in MVC nicht nur die Aufgabe, die Programdaten, Logik und Zustände zu verwalten, sondern bietet auch eine Schnittstelle an, um auf diese Daten zuzugreifen. In diesem Kapitel wird die Interne Funktionalität des Modells genauer erklärt.

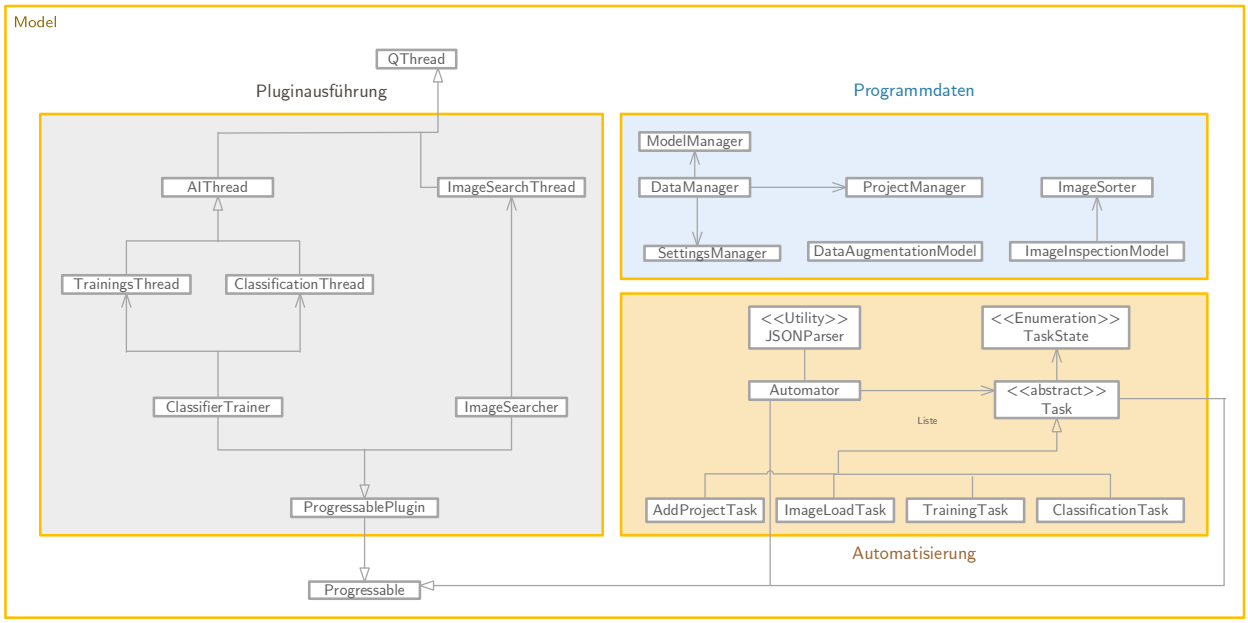


Abbildung 3 : Aufbau des Model

2.1 Entwurf

〉 Programmdaten

Die Klassen in dieser Kategorie verwalten verschiedene private Attribute zu Projektdaten, Modelldaten und globale Einstellungen. Weiter werden Informationen zur *Data Augmentation,* sowie für die Bildvorschau verwaltet. Der Zugriff auf Attribute erfolgt über Getter und Daten werden durch bestimmte Methoden gesetzt. Die Methoden der Klassen `*Manager` sind alle über die Fassade `DataManager` erreichbar.

〉 Pluginausführung

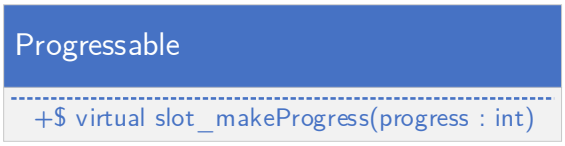
Die Klassen dieser Kategorie sind aufgeteilt in *Progressables* und *Threads*. Die *Threads* ermöglichen eine parallelisierte Ausführung der AI-/Bildsuchplugins. Die *Progressables* ermöglichen es den Plugins ihren Fortschritt mitzuteilen und stellen sicher, dass Informationen (wie das Beenden eines Plugins, beziehungsweise dessen möglichen Ausfall) durch Signale an den Controller weitergeleitet werden.

〉 Automatisierung

Die Klassen dieser Kategorie erfüllen das Wunschkriterium „Batchverarbeitung“ aus dem Pflichtenheft. Die Automatisierung wird durch Parsen einer JSON-Datei ermöglicht. Es werden aus den Inhalten der Datei entsprechende Argumente für den Task entnommen, um diese zu erledigen. Der *Automator* verwaltet hierbei die einzelnen Tasks.

2.2 Klassen

2.2.1 Progressable



Klassenbeschreibung

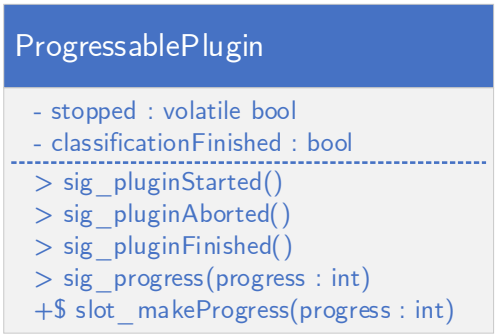
〉 Stellt einen Slot bereit, um Fortschritt eines Prozesses entgegenzunehmen.

Methoden

`virtual slot_makeProgress(progress : int)`

•Wird aufgerufen, wenn der Prozess Fortschritt übermittelt.

2.2.2 ProgressablePlugin



Klassenbeschreibung

〉 Die Klasse beinhaltet gemeinsame Eigenschaften von `ImageLoader` und `ClassifierTrainer`.

〉 Die Klasse erweitert `Progressable`.

Methoden

sig_pluginStarted()

- Das Signal wird ausgelöst, wenn der Algorithmus startet.

sig_pluginAborted()

- Das Signal wird ausgelöst, wenn das Plugin abbricht.

sig_pluginFinished()

- Das Signal wird ausgelöst, wenn das Plugin fertig ist.

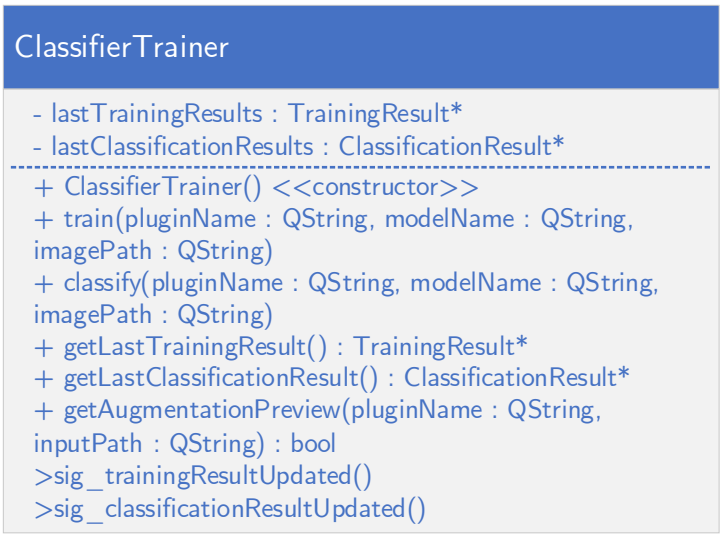
sig_progress(progress : int)

- Das Signal wird benutzt, um den Fortschritt des Threads an die GUI weiterzuleiten.

slot_makeProgress(progress : int)

- Der Slot wird von den Plugins aufgerufen wenn Fortschritt gemacht wird.

2.2.3 ClassifierTrainer



Klassenbeschreibung

〉 Die Klasse ist dafür zuständig, einen `ClassificationThread` oder einen `TrainingThread` zu erstellen, welcher mittels Signals den Klassifizierungs-/Trainingsplugin seinen Fortschritt kommuniziert

〉 Die Klasse erbt von `ProgressablePlugin`, benutzt also insbesondere alle darin definierten Signals und Slots.

Konstruktoren

`ClassifierTrainer()`

- Erstellt eine Instanz von ClassifierTrainer.

Methoden

`train(pluginName : QString, modelName : QString, imagePath : QString)`

- Methode wird verwendet um ein Training mit dem ClassificationAndTrainingthread und den gegeben Argumenten zu starten.

`classify(pluginName : QString, modelName : QString, imagePath : QString)`

- Methode wird verwendet um eine Klassifizierung mit dem ClassificationAndTrainingthread und den gegeben Argumenten zu starten.

`getLastTrainingResult() : TrainingResult*`

- Gibt das zuletzt bekommene Trainingsergebnis zurück.

`getLastClassificationResult() : ClassificationResult*`

- Gibt das zuletzt bekommene Klassifizierungsergebnis zurück.

`getAugmentationPreview(pluginName : QString, inputPath : QString) : bool`

- Gibt eine Vorschau der DataAugmentation zurück.

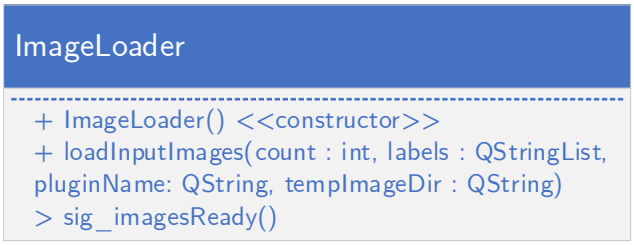
`sig_trainingResultUpdated()`

- Das Signal wird ausgelöst, wenn der aktuelle Trainingsvorgang beendet ist und (neue) Ergebnisse vorliegen.
- Ist mit dem gleichnamigen Slot im AIController verbunden.

`sig_classificationResultUpdated()`

- Das Signal wird ausgelöst, wenn der aktuelle Klassifizierungsvorgang beendet ist und (neue) Ergebnisse vorliegen.
- Ist mit dem gleichnamigen Slot im AIController verbunden.

2.2.4 ImageLoader



Klassenbeschreibung

- Die Klasse ist dafür zuständig einen `ImageSearchThread` zu erstellen, welcher mittels Signals den Bildsammlerplugin Fortschritt kommuniziert
- Die Klasse erbt von `ProgressablePlugin`, benutzt also insbesondere alle darin definierten Signals und Slots.

Attribute

- Referenz zu `ImageSearchThread`

Konstruktoren

`ImageLoader()`

- Erstellt den ImageLoader.

Methoden

`loadInputPictures(count : int, labels : QStringList, pluginName : QString, tempImageDir : QString)`

- Methode wird verwendet um den ImageSearchThread mit den gegeben Argumenten zu starten.

`sig_imagesReady()`

- Das Signal wird ausgelöst, wenn Bildersuche beendet ist die Bilder weiterverarbeitet werden können.

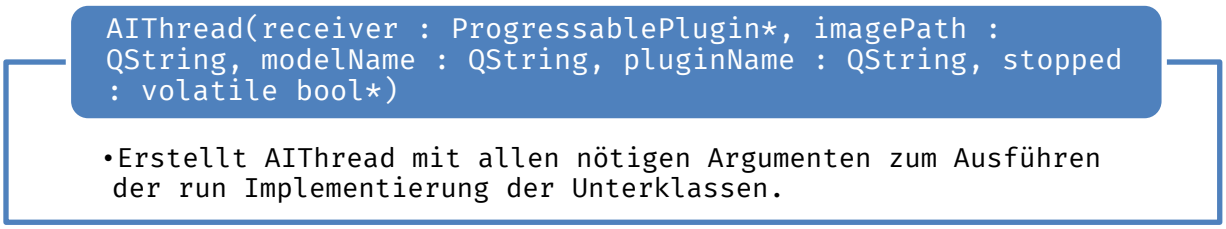
2.2.5 AIThread

Klassenbeschreibung

〉 Die abstrakte Klasse ist eine Oberklasse um die Klassifizierungs-/Trainingsplugins parallelisiert auszuführen.

〉 Die Klasse erbt von `QThread`.

Konstruktoren



2.2.6 TrainingsThread

Klassenbeschreibung

〉 Die Klasse ermöglicht eine parallelisierte Ausführung des Trainings.

〉 Die Klasse erbt von `AIThread` und implementiert alle virtuellen Methoden.

Konstruktoren

```
TrainingsThread(receiver : ProgressablePlugin*, imagePath :
QString, modelName : QString, pluginName : QString)
```

•Erstellt TrainingsThread mit allen nötigen Argumenten zum Ausführen der run Methode.

Methoden

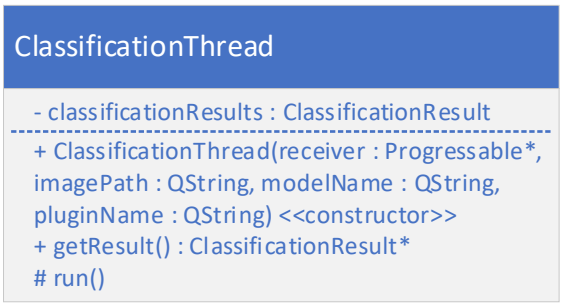
```
getResult() : TrainingsResult
```

•Getter für Trainingsergebnisse.

```
run()
```

•Wird verwendet um das Training mittels des ClassificationPluginManagers zu starten.

2.2.7 ClassificationThread



Klassenbeschreibung

〉 Die Klasse ist ermöglicht eine parallelisierte Ausführung der Klassifizierung.

〉 Die Klasse erbt von `AIThread` und implementiert alle virtuellen Methoden.

Konstruktoren

```
ClassificationThread(receiver : Progressable*,
imagePath : QString, modelName : QString,
pluginName : QString, stopped : volatile bool*)
```

•Erstellt ClassificationThread mit allen nötigen Argumenten zum Ausführen der run Methode.

Methoden

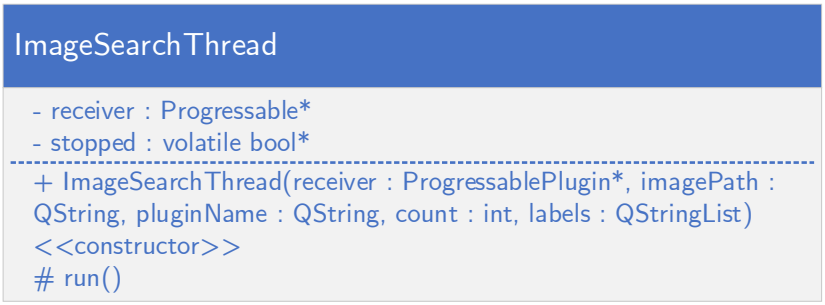
getResult() : ClassificationResult

- Getter für Klassifikationsergebnisse.

run()

- Wird verwendet um die Klassifizierung mittels des ClassificationPluginManagers zu starten.

2.2.8 ImageSearchThread



Klassenbeschreibung

〉 Die Klasse ermöglicht eine parallelisierte Ausführung der Bildsammlung.

〉 Die Klasse erbt von `QThread` und implementiert die `run` Methode.

Konstruktoren

ImageSearchThread(receiver : Progressable*, imagePath : QString, modelName : QString, pluginName : QString, count : int, labels : QStringList)

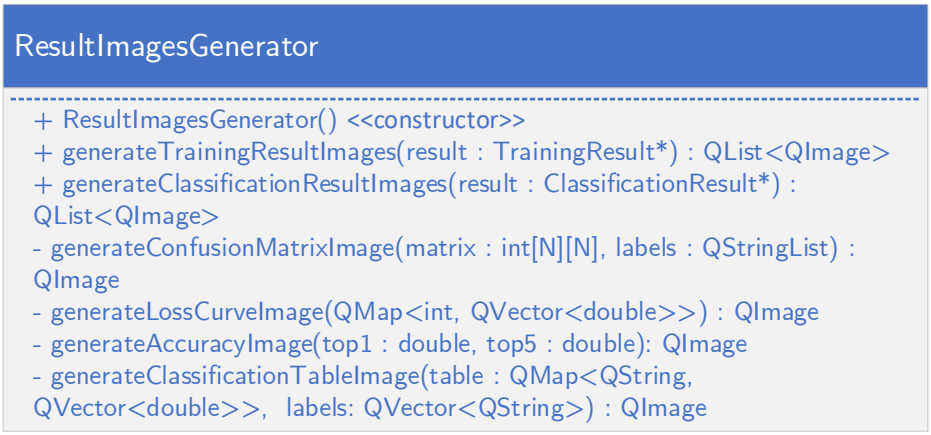
- Erstellt ImageSearchThread mit allen nötigen Argumenten zum Ausführen der run Methode

Methoden

run()

- Wird verwendet um die Bildsuche mittels des PictureLoaderPluginManagers zu starten.

2.2.9 ResultImagesGenerator



Klassenbeschreibung

〉 Die Klasse `ResultImagesGenerator` ist für die Generierung der Darstellungen der Trainings- und Klassifizierungsergebnissen verantwortlich

Konstruktoren

`ResultImagesGenerator()`

- `Instaziiert ein neues Objekt der ResultImageGenerator Klasse`

Methoden

`generateTrainingResultImages(result : TrainingResult*) : QList<QImage>`

- Diese Methode leitet die Generierung der Bilder zur Darstellung der einzelnen Trainingsergebnisse ein.

`generateClassificationResultImages(result : ClassificationResult*) : QList<QImage>`

- Diese Methode leitet die Generierung der Bilder zur Darstellung der einzelnen Klassifizierungsergebnisse ein.

`generateConfusionMatrixImage(matrix : int[N][N], labels : QStringList) : QImage`

- Generiert aus den gelieferten Daten eine Konfusionsmatrix und gibt diese als QImage zurück.

`generateLossCurveImage(QMap<int, QVector<double>>) : QImage`

- Generiert aus den gelieferten Daten eine Verlustkurve und gibt diese als QImage zurück.

`generateAccuracyImage(top1 : double, top5 : double): QImage`

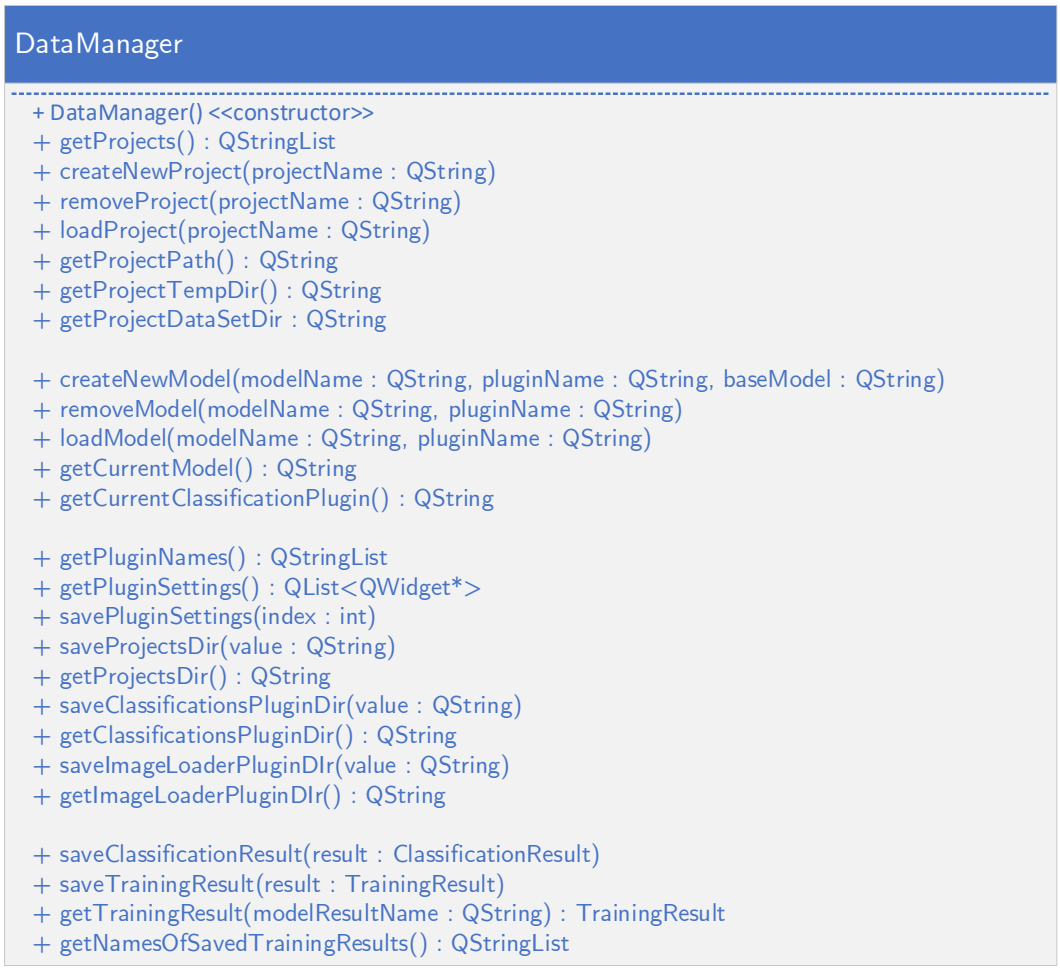
- Generiert aus den gelieferten Top1 und Top5 Genauigkeitswerten eine Grafik und gibt diese als QImage zurück.

`generateClassificationTableImage(table : QMap<QString, QVector<double>>, labels : QVector<QString>) : QImage`

- Generiert eine Tabelle für die klassifizierten Bilder und gibt sie als QImage zurück.

2.2.10 DataManager

Klassenbeschreibung



- Die Klasse ist der zentrale Ansprechpunkt für alle Datenzugriffe seitens der Controller.
- Die Klasse stellt eine Fassade für die Klassen `SettingsManager`, `ModelManager` und `ProjectManager` dar.

Attribute

- Referenz zu `ProjectManager`
- Referenz zu `ModelManager`
- Referenz zu `SettingsManager`

Konstruktoren

`DataManager()`

- `Erstellt DataManager ohne bestimmte Daten.`

Methoden


getProjects() : QStringList
•Gibt eine Liste aller dem Programm bekannten Projekte zurück.
createNewProject(projectName : QString)
•Ruft die entsprechende Methode in ProjectManager auf.
removeProject(projectName : QString)
•Ruft die entsprechende Methode in ProjectManager auf.
loadProject(projectName : QString)
•Ruft die entsprechende Methode in ProjectManager auf.
getProjectDir() : QString
•Ruft die entsprechende Methode in ProjectManager auf.
getProjectTempDir() : QString
•Ruft die entsprechende Methode in ProjectManager auf.
getProjectDataSetDir() : QString
•Ruft die entsprechende Methode in ProjectManager auf.
createNewModel(modelName : QString, pluginName : QString, baseModel : QString)
•Ruft die entsprechende Methode in ModelManager auf.
removeModel(modelName: QString, pluginName: QString)
•Ruft die entsprechende Methode in ModelManager auf.

loadModel(modelName : QString, pluginName : QString)

- Ruft die entsprechende Methode in ModelManager auf.

getCurrentModel() : QString

- Ruft die entsprechende Methode in ModelManager auf.

getCurrentPluginClassificationPlugin() : QString

- Ruft die entsprechende Methode in ModelManager auf.

getPluginNames() : QStringList

- Ruft die entsprechende Methode in SettingsManager auf.

getPluginSettings() : QList<QWidget*>

- Ruft die entsprechende Methode in SettingsManager auf.

savePluginSettings(index : int)

- Ruft die entsprechende Methode in SettingsManager auf.

saveProjectsDir(value : QString)

- Ruft die entsprechende Methode in SettingsManager auf.

getProjectsDir() : QString

- Ruft die entsprechende Methode in SettingsManager auf.

saveClassificationsPluginDir(value : QString)

- Ruft die entsprechende Methode in SettingsManager auf.

getClassificationsPluginDir() : QString

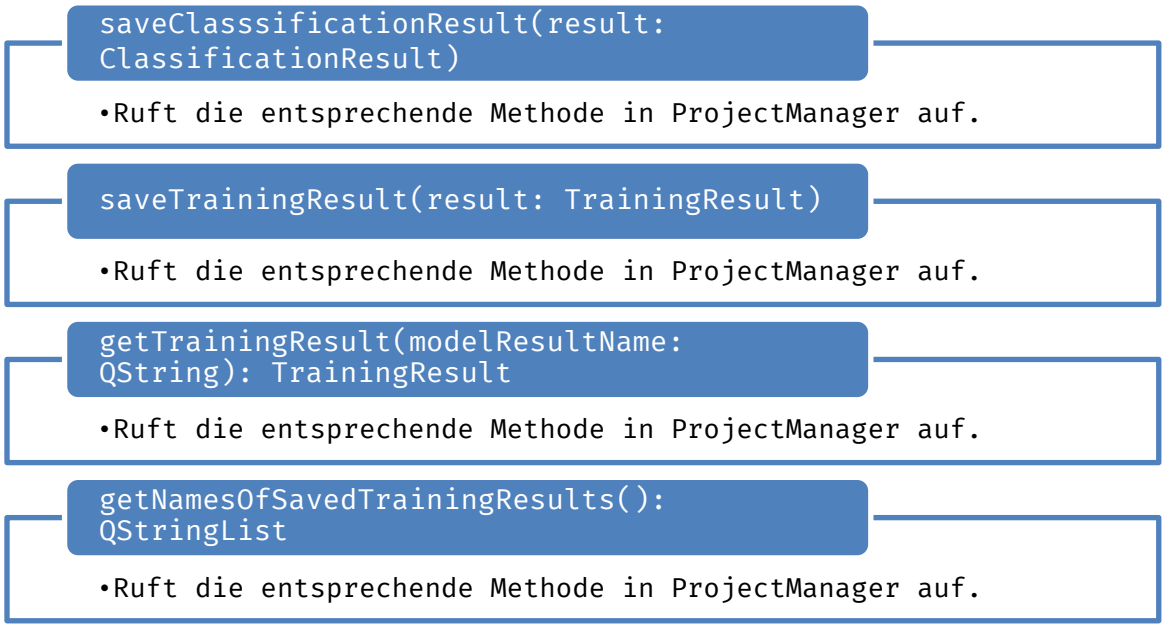
- Ruft die entsprechende Methode in SettingsManager auf.

saveImageLoaderPluginDIr(value : QString)

- Ruft die entsprechende Methode in SettingsManager auf.

getImageLoaderPluginDIr() : QString

- Ruft die entsprechende Methode in SettingsManager auf.


saveClasssificationResult(result: ClassificationResult)
•Ruft die entsprechende Methode in ProjectManager auf.
saveTrainingResult(result: TrainingResult)
•Ruft die entsprechende Methode in ProjectManager auf.
getTrainingResult(modelResultName: QString): TrainingResult
•Ruft die entsprechende Methode in ProjectManager auf.
getNamesOfSavedTrainingResults(): QStringList
•Ruft die entsprechende Methode in ProjectManager auf.

2.2.11 SettingsManager

Klassenbeschreibung

〉 Die Klasse ist zuständig für das Speichern globaler Einstellungen.

Konstruktoren

```
SettingsManager(
classificationPluginManager : ClassificationPluginManager*,
imageLoaderPluginManager : ImageLoaderPluginManager*)
```

- Erstellt SettingsManager mit den gewünschten Pfadeinstellungen.

Methoden

getPluginNames() : QStringList

- Liefert die Namen aller Plugins in einer Liste .

getPluginSettings() : QList<QWidget*>

- Gibt die Einstellungswidget von allen Plugins zurück.

savePluginSettings(index : int)

- Speichert die Einstellungen des Plugins mit dem zugehörigen Listeneintrag.

saveProjectsDir(value : QString)

- Speichert das globale Projektverzeichnis.

getProjectsDir() : QString

- Gibt das globale Projektverzeichnis zurück.

saveClassificationsPluginDir(value : QString)

- Speichert den Ablageort für die Klassifikationsplugins.

getClassificationsPluginDir() : QString

- Setzt den Ablageort für die Klassifikationsplugins.

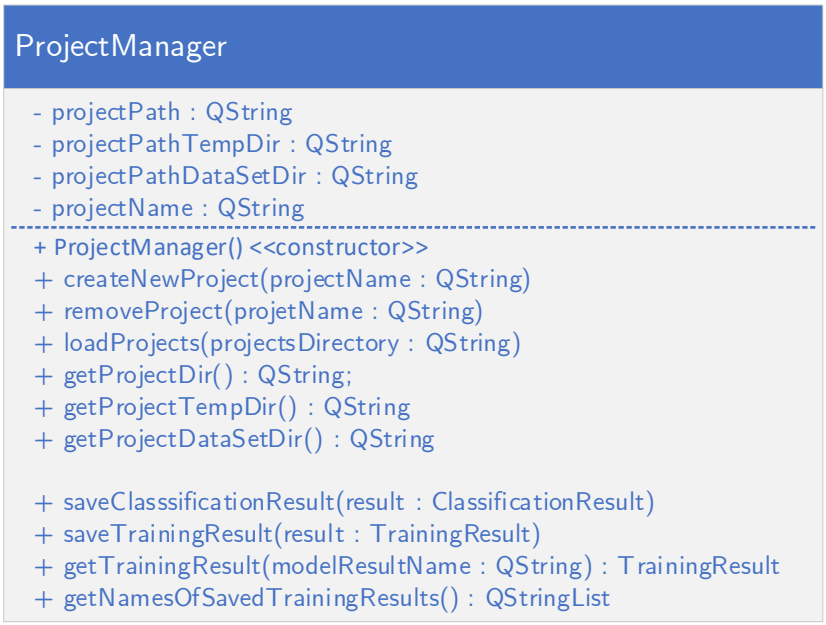
saveImageLoaderPluginDir(value : QString)

- Speichert den Ablageort für die Bildsammlerplugins.

getImageLoaderPluginDir() : QString

- Gibt den Ablageort für die Bildsammlerplugins zurück.

2.2.12 ProjectManager



Klassenbeschreibung

〉 Die Klasse ist zuständig für die Verwaltung der Projektdaten

Konstruktoren

ProjectManager()

- Erstellt ProjectManager ohne bestimmte Daten

Methoden

createNewProject(projectName : QString)

- Erstellt ein neues Projekt mit gegebenem Namen.

removeProject(projectName : QString)

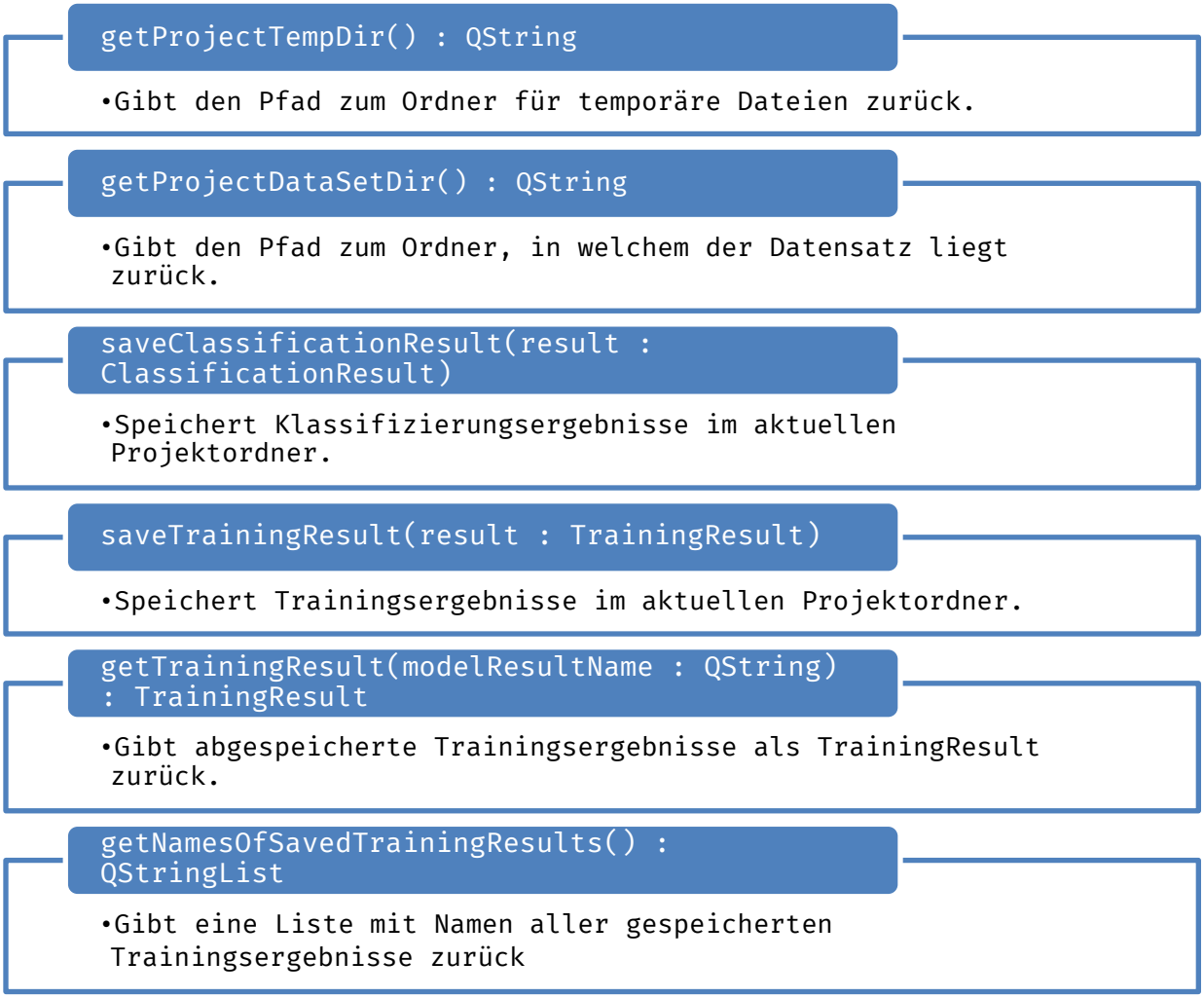
- Entfernt das Projekt mit gegebenem Namen.

loadProjects(projectDir : QString)

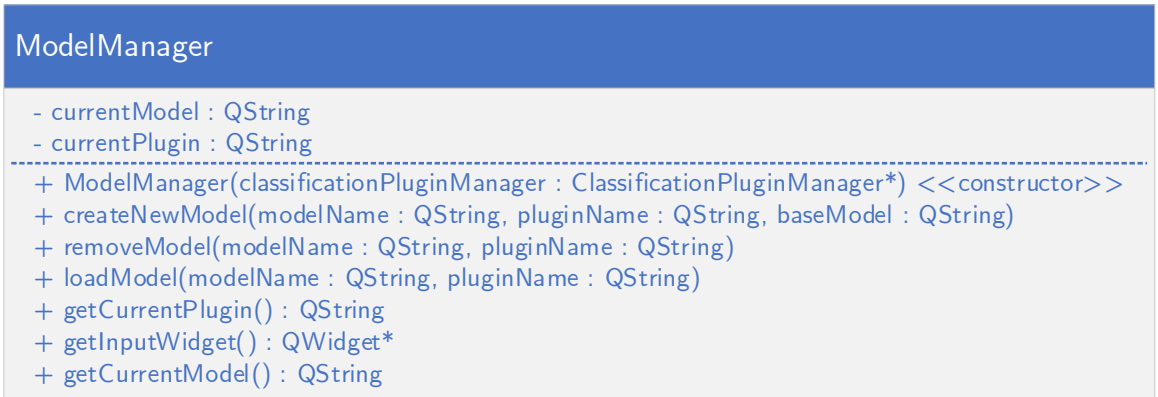
- Lädt Projekte in gegebenem Ordner.

getProjectDir() : QString

- Gibt den Pfad zum aktuellen Projektordner zurück.



2.2.13 ModelManager



Klassenbeschreibung

〉 Die Klasse ist zuständig für die Verwaltung des Machine Learning Models.

Konstruktoren

`ModelManager(classificationPluginManager: ClassificationPluginManager*)`

- Erstellt ModelManager mit Refernz auf ClassificationPluginManager.

Methoden

`createNewModel(modelName : QString, pluginName : QString, baseModel : QString)`

- Erstellt ein neues Modell für das gegebene Plugin auf dem angegebenen Basismodell

`removeModel(modelName : QString, pluginName : QString)`

- Löscht das angegebene Modell für das Plugin.

`loadModel(modelName : QString, pluginName : QString)`

- Läd das angegebene Modell zur weiteren Verwendung.

`getCurrentPlugin() : QString`

- Gibt das gerade ausgewählte Plugin zurück.

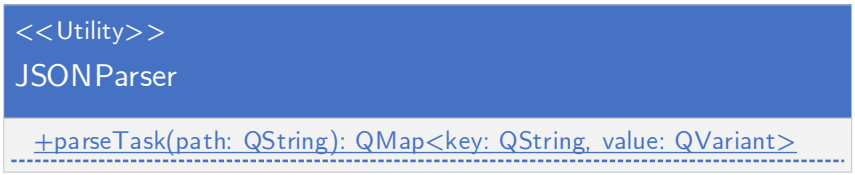
`getInputWidget() : QWidget*`

- Gibt das Eingabe-Widget des aktuell ausgewählten Plugins zurück.

`getCurrentModel() : QString`

- Gibt den Namen des gerade ausgewählten Models zurück.

2.2.14 JSONParser



Klassenbeschreibung

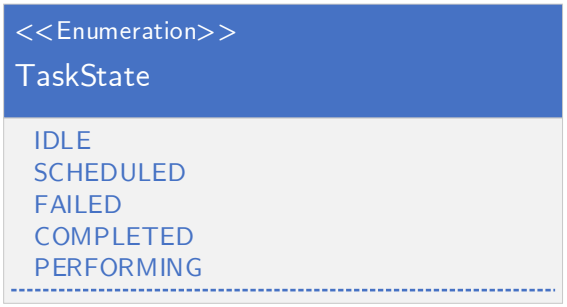
〉 Die Klasse ist eine Utility-Klasse, welche JSON-Dateien in eine Folge Schlüssel-Wert-Paare umwandelt.

Methoden

```
parseTask(path : QString) : QMap<key :
QString, value : QVariant>
```

- Öffnet die Datei in path und gibt deren Inhalt als QMap zurück.

2.2.15 TaskState



Klassenbeschreibung

〉 Das `Enum` beschreibt mögliche Zustände, in denen sich ein `Task` befinden kann.

2.2.16 Task

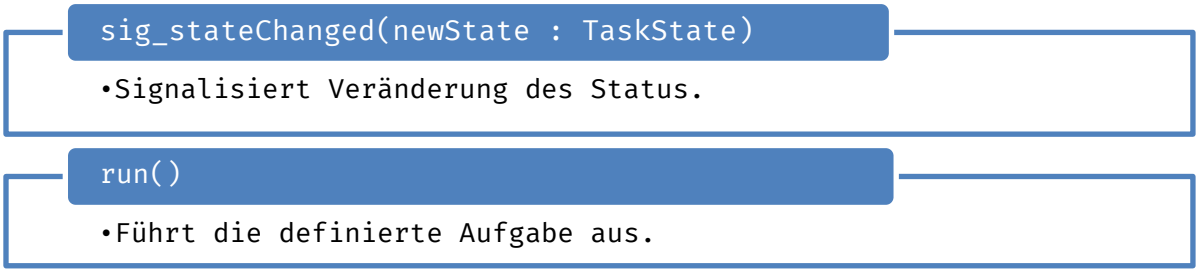
Klassenbeschreibung

- Die Klasse ist eine Oberklasse für alle Taskklassen und hält gemeinsame Attribute / Methoden.
- Die Klasse erbt von `Progressable` um Fortschritt anzuzeigen

Attribute

- Referenz zum `DataManager`

Methoden



2.2.17 AddProjectTask

Klassenbeschreibung

- Die Klasse erbt von `Task` und enthält dementsprechend alle Attribute und Methoden aus Task.
- Beim Aufrufen von `run()` wird ein Projekt mittels der in `QMap` gegebenen Informationen hinzugefügt.

2.2.18 ImageLoadTask

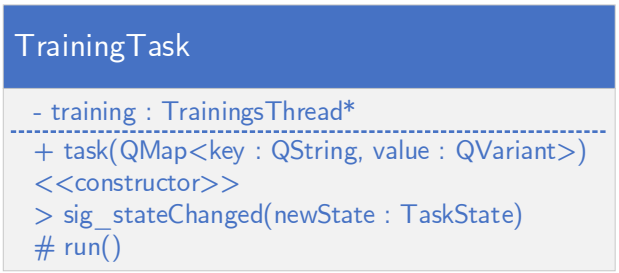
Klassenbeschreibung

- Die Klasse erbt von `Task` und enthält dementsprechend alle Attribute und Methoden aus Task.
- Zudem wird bei Aufrufen von `run()` ein Bildladevorgang mittels der in `QMap` gegebenen Information durchgeführt.

Attribute

- Referenz auf einen `ImageSearchThread`, welcher mittels der Informationen in `QMap` erzeugt wurde.

2.2.19 TrainingTask



Klassenbeschreibung

- Die Klasse erbt von `Task` und enthält dementsprechend alle Attribute und Methoden aus `Task`.
- Zudem wird bei Aufrufen von `run()` ein Trainingsvorgang mittels der in `QMap` gegebenen Information durchgeführt.

Attribute

- Referenz auf einen `TrainingsThread`, welcher mittels der Informationen in `QMap` erzeugt wurde.

2.2.20 ClassificationTask

Klassenbeschreibung

- Die Klasse erbt von `Task` und enthält dementsprechend alle Attribute und Methoden aus `Task`.
- Zudem wird bei Aufrufen von `run()` ein Klassifizierungsvorgang mittels der in `QMap` gegebenen Information durchgeführt.

Attribute

- Referenz auf einen `ClassificationThread`, welcher mittels der Informationen in `QMap` erzeugt wurde.

2.2.21 Automator

Klassenbeschreibung

- Die Klasse verwaltet die `Tasks`, ist also für das Hinzufügen, Entfernen und Starten der `Tasks` verantwortlich.
- Die Klasse erbt von `Progressable` um Fortschritt anzuzeigen.

Attribute

- Referenz auf `DataManager`.

Methoden

sig_taskUpdate()

- Wird ausgelöst, wenn sich ein Task seinen Status verändert.

slot_taskUpdated()

- Wird bei aktualisierung

performTasks()

- Startet Ausführung der in der Liste enthaltenen Tasks.

addTasks(path : QString)

- Liest eine Datei in gegebenen Pfad und fügt die dort definierten Tasks hinzu (in Idlelliste).

remove(taskNum : int)

- Entfernt den Task mit gegebenem Index aus der Idleliste.

setIdle(taskNum : int)

- Verschiebt den entsprechenden Eintrag von Sheduelliste ans Ende von Idleliste.

shedule(taskNum : int)

- Verschiebt den entsprechenden Eintrag von Idleliste ans Ende von Sheduelliste.

getIdleTasks() : QList <tasks : QPair<Name : QString, state : TaskState>>

- Gibt eine Liste der untätigen Tasks zurück.

getScheduledTasks() : QList <tasks : QPair<Name : QString, state : TaskState>>

- Gibt eine Liste der zur Ausführung geplanten Tasks zurück.

2.3 Datenspeicherung

Damit die zu speichernden Daten in einem maschinenlesbaren Format gespeichert und für die weitere Verwendung auch geladen werden können, muss ein einheitliches Format vorhanden sein. Gespeichert werden Projektdateien, Modellzugehörigkeitsdateien, Trainingsdurchlaufdateien, Klassifizierungsdurchlaufdateien und die Einstellungsdatei. Die Dateien werden mithilfe von `QSettings` persistent gespeichert. Dies ermöglicht eine plattformunabhängige Abspeicherung dieser Daten. Die Trainingsdurchlaufdateien und Klassifizierungsdurchlaufdateien zum Speichern der Ergebnisse beinhalten dementsprechend neben einem Identifier lediglich den Pfad zu den zum Durchlauf gehörenden Ergebnissen. Des Weiteren gibt es den Bilderdatensatz, Vorschaubilder für die Data Augmentation und die Trainings- und Klassifizierungsergebnisse. Da man sich dem Pflichtenheft entsprechend bei der Abspeicherung des Datensatzes auf das ImageNet Format für den Datensatz festgelegt hat, wird dieses zum Annotieren des Datensatzes verwendet. Temporäre Bilder wie z.B. Vorschaubilder werden dagegen in einem externen Ordner innerhalb des zugehörigen Projektverzeichnisses gespeichert. Da man Ergebnisse vergleichen will, müssen diese aus einem Trainingsobjekt oder Klassifizierungsobjekt entsprechend ausgelesen und abgespeichert werden. Im Folgendem der schematische Aufbau der anderen Dateien.

Aufbau einer Projektdatei:

```
{
    "MetaData": {
        "projectName":"Project 1",
        "projectDir": "C:/User/Projects/",
        "datasetDirName":  "data",
        "tempDirName":     "temp"
        }
}
```

Aufbau der Modellzugehörigkeitsdatei:

```
{
    "Model1":   {
        "ModelName": "ExampleModel",
        "PluginName": "MMClassification",
        }
}
```

Aufbau der Einstellungsdatei:

```
{
    "ApplicationSettings": {
        "StandardWorkingPath":
        "C:/Programme/RapidClassificationTrainer/WorkingDirectory",

        "classificationPluginPath":
        "C:/Programme/RapidClassificationTrainer/ClassificationPlugins",

        "imageLoaderPluginPath":
        "C:/Programme/RapidClassificationTrainer/imageLoaderPlugins"
    }
}
```

3 View

Die Präsentation (englisch: View) ist im MVC-Architekturstil dafür verantwortlich die Modell-Daten anzuzeigen und Nutzereingaben entgegenzunehmen. In der hier gewählten Variante von MVC veranlasst der Controller die Änderung der Anzeige. Genauere Entwurfsdetails werden in diesem Kapitel vorgestellt.

3.1 Entwurf

Im MVC ist der View für die Datendarstellung verantwortlich. Es werden Objekte und Informationen aus dem Modell in einem Benutzer leserlichen Format wiedergegeben. Des Weiteren bietet es dem Nutzer die Möglichkeit Eingaben zu verfassen, so kann dieser mit den Daten interagieren und sie beschränkt bearbeiten. Hierbei ist wichtig zu beachten, dass der View keinerlei Anwendungslogik enthält, diese wird nämlich in die Subsysteme für das Modell und den Controller ausgelagert.

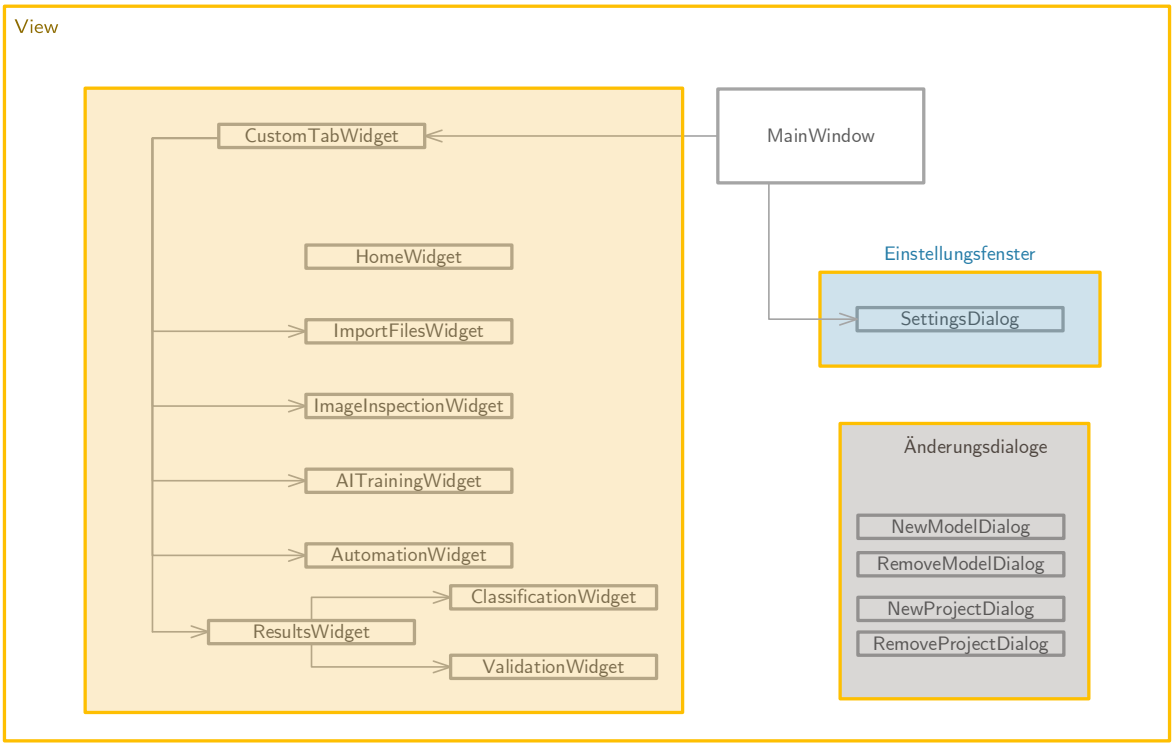


Abbildung 4 : View Übersicht

Das Einstellungsfenster sowohl auch die Fenster für Projekt und Modell Operationen sind nicht direkt wie die anderen Informationsanzeigen in der Hauptansicht verankert. Diese Fenster können nämlich über Schaltflächen auf der Hauptansicht separat geöffnet werden. Kommunikation zwischen den Fenstern und der Anwendung verläuft, wie bei den Informationsanzeigen, über den Controller.

Anmerkung: Die Klassen innerhalb des View Pakets deren Name auf dem Suffix „Widget“ oder „Dialog“ endet, erweitern jeweils die Klasse `QWidget` und `QDialog`. So erweitert beispielsweise die Klasse `SettingsDialog` die Klasse `QDialog`, und `HomeWidget` erweitert entsprechend `QWidget`. Dieser Zusammenhang wurde in den folgenden Diagrammen aus Übersichtlichkeitsgründen nicht explizit dargestellt und geht anhand dieser Anmerkung und der Benennung der Klassen implizit vor.

MainWindow

- Für die Navigation innerhalb des Hauptfensters wird eine Tableiste (*tabBar*) zur Verfügung gestellt die die Funktionalität als auch die Informationsanzeigen in 7 verschiedene Tabs (*tabWidget*) unterteilt (Abb. 3).
- Innerhalb der Tab Leiste sind die einzelnen Widgets mit Home, Import Files, Image Inspection, AI Training, Input Images, Results und Automation betitelt. Diese bieten dem Benutzer die Möglichkeit, Eingaben zu verfassen und diese einzusehen bzw. abzuändern.

Informationsanzeigen

- Im Home Widget werden, durch ein listWidget, die Projekte angezeigt, diese können von dem Benutzer erstellt, gelöscht und ausgewählt werden (siehe Erstellen/Löschen Fenster). In der Tableiste direkt daneben bietet das Import Files Widget die Möglichkeit Eingabedaten zu definieren bzw. anzufordern sowohl auch einen ladefortschritt einzusehen (*progressBar*). Die Eingabedaten werden unter anderem mittels QSpinBox, QSlider und QComboBox durch den Benutzer übergeben. Da die diese Art von Eingabefeld den Freiheitsgrad der Eingabe beschränken können hier ungültige Eingaben auf Seiten des Benutzers verhindert werden.
- Nach erfolgreicher Angabe der Eingabedaten werden entsprechende Bilder im Nächsten Widget dargestellt. Mit Hilfe eines Scrollbalken kann die Ansicht verschoben werden und der Benutzer ist hier nochmal in der Lage die angezeigten Eingabebilder abzuändern. Das AI Training Widget ist dafür zuständig dem Nutzer die Möglichkeit zu bieten Trainingsparameter anzupassen und schließlich auch den Fortschritt des aktuell laufenden Trainings anzuzeigen.
- Nach erfolgreicher Ausführung eines Trainingsprozess werden entsprechende Ergebnisse im Ergebnisse Tab dem Benutzer angezeigt. Des Weiteren bietet dann das Input Images Tab noch die Möglichkeit eine Klassifikation zu parametrisieren und schließlich auch auszuführen. Die Ergebnisse dessen werden dann wiederrum im Ergebnisse Tab erscheinen.
- Letztlich ermöglicht das Automation Widget dem Benutzer bestimmte Programmabläufe in einzelne Tasks auszulagern. Dies bietet eine zweite Möglichkeit die Funktionalität des Programms anzusteuern, in der insbesondere die Notwendigkeit alle Eingabedaten Manuell verfassen zu müssen entfällt.

Änderungsdialoge

- Das Erstellen Fenster öffnet sich, um dem Benutzer relevante Einstellungen in Bezug auf die Erstellung eines neuen Projekts, oder Modells anzuzeigen. Für ein neues Projekt kann hier der Name und der Ordner parametrisiert werden, Für Modelle kann sowohl der Name als auch noch Plugin und Basis eingestellt werden.
- Das Löschen Fenster öffnet sich, um das Löschen eines Projekts oder Modells nochmal durch den Benutzer bestätigen zu lassen. Auf diese Weise können ungewollte oder versehentliche Eingaben auf Seiten des Benutzers teils verhindert werden.

Abbildung 5.1 : Aufbau des Hauptfensters

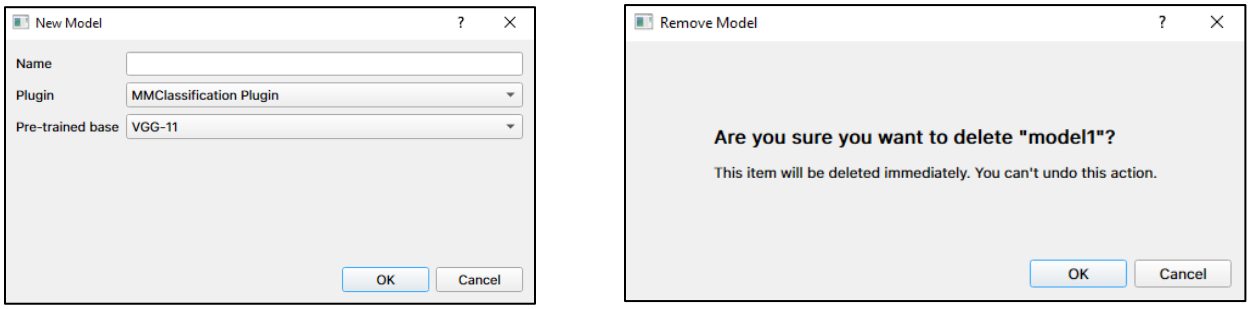
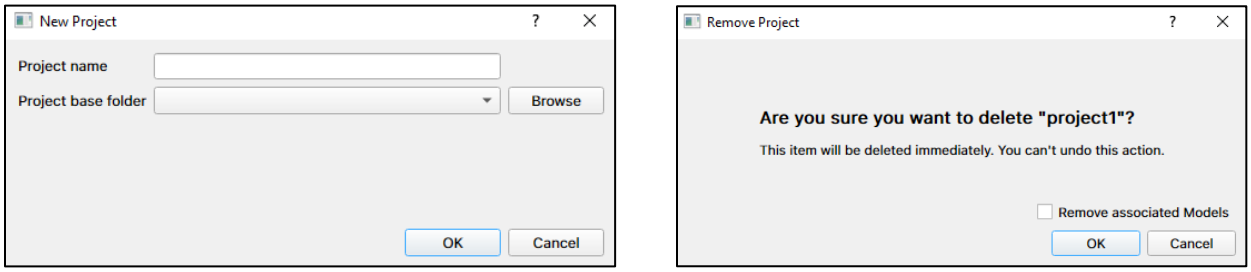


Abbildung 3.2 : Modell Erstellungs- und Löschungsdialog



〉 *Siehe Pflichtenheft für Abbildungen der anderen Widgets.*

Abbildung 3.3 : Projekt Erstellungs- und Löschungsdialog

3.2 Klassen

3.2.1 MainWindow

Klassenbeschreibung

〉 Das Hauptfenster, auf welchem alle referenzierten Widgets verankert sind.

Konstruktoren

`MainWindow(parent : QWidget*)`

- Konstruktor, welcher zur Erstellung eines QWindows verwendet wird und alle Signale mit zugehörigen Slots verknüpft.

Methoden

`sig_openSettings()`

- Das Signal wird ausgelöst wenn auf das Zahnrad gecklickt wird.
- Das Signal löst den gleichnamigen Slot im SettingsController aus.

3.2.2 HomeWidget

Klassenbeschreibung

〉 Das `HomeWidget`, ermöglicht die Auswahl eines gewünschten Projekts, bzw. auch das Erstellen und Löschen von Projekten.

Konstruktoren

HomeWidget(parent : QWidget*, projects: QStringList)

- Instantiiert das Widget.

Methoden

setProjects(projects : QStringList)

- Setzt die anzuzeigende Projekt Liste.

sig_newProject()

- Das Signal wird ausgelöst, wenn "New Project" gelickt wird.
- Das Signal löst den gleichnamigen Slot im ProjectController aus.

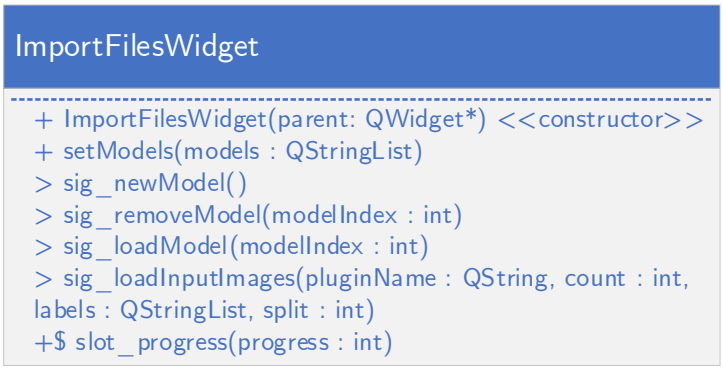
sig_removeProject()

- Das Signal wird ausgelöst, wenn "Remove" gelickt wird.
- Das Signal löst den gleichnamigen Slot im ProjectController aus.

sig_openProject()

- Das Signal wird ausgelöst, wenn "Open" gelickt wird.
- Das Signal löst den gleichnamigen Slot im Controller aus.

3.2.3 ImportFilesWidget



Klassenbeschreibung

〉 Das `ImportFilesWidget`, ermöglicht die Definition von Trainings- und Validierungsbildern sowohl auch das Laden von einem gewünschten Modell.

Konstruktoren

ImportFilesWidget(parent : QWidget*, projects: QStringList)

- Instantiiert das Widget.

Methoden

setModels(models : QStringList)

- Setzt die anzuzeigende Model Liste.

sig_newModel()

- Das Signal wird ausgelöst, wenn auf "Add" geclickt wird.
- Das Signal löst den gleichnamigen Slot im ModelController aus.

sig_removeModel()

- Das Signal wird ausgelöst, wenn auf "Remove" geclickt wird.
- Das Signal löst den gleichnamigen Slot im ModelController aus.

sig_loadModel()

- Das Signal wird ausgelöst, wenn auf "Load" geclickt wird.
- Das Signal löst den gleichnamigen Slot im Controller aus.

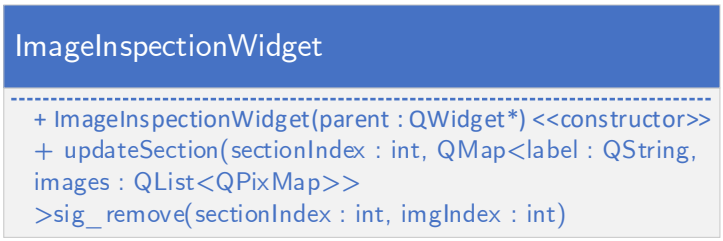
sig_LoadInputPictures(pluginName : QString, count : int, labels : QStringList))

- Das Signal wird ausgelöst, wenn auf "Load Images" geclickt wird.
- Der Slot ist mit dem gleichnamigen Signal in ImageController verbunden.

slot_progress(progress : int)

- Dieser Slot wird von dem verwendeten Bildsammlerplugin aufgerufen, wenn ein Fortschritt im Ladeprozess vermeldet wird.
- Der Slot ist mit dem gleichnamigen Signal des ImageLoader verbunden.

3.2.4 ImageInspectionWidget



Klassenbeschreibung

〉 Das `ImageInspectionWidget` zeigt die heruntergeladenen Trainings- und Validierungsbilder an. Der Benutzer kann hier Bilder verschieben und löschen.

Konstruktoren

ImageInspectionWidget(parent : QWidget*)

- Instantiiert das Widget.

Methoden

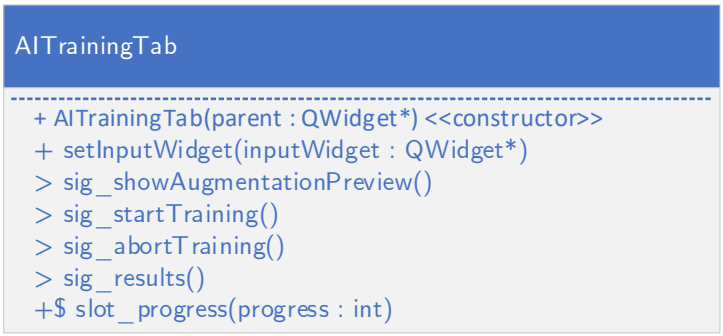
updateSection(sectionIndex : int, QMap<label : QString, images : QList<QPixMap>>

- Setzt die anzuzeigenden Trainings- und Validierungsbilder des Datensatzes und der neu durch das Bildsammlerplugin heruntergelandenen Bilder

sig_remove(sectionIndex : int, imgIndex : int)

- Das Signal wird ausgelöst, wenn auf "Remove" geclickt wird.
- Das Signal löst den gleichnamigen Slot im ImageController aus.

3.2.5 AITrainingWidget



Klassenbeschreibung

〉 Das `AITrainingWidget`, beinhaltet die Parametrisierung des Modelltrainings und zeigt Information bezüglich des Augmentierens der Eingabedaten an.

Konstruktoren

`AITrainingTab(parent : QWidget*)`

- Instantiiert das Widget.

Methoden

sig_startTraining()

- Das Signal wird ausgelöst, wenn auf "Start" geclickt wird.
- Das Signal löst den gleichnamigen Slot im AIController controller aus.

sig_abortTraining()

- Das Signal wird ausgelöst, wenn auf "Cancel" geclickt wird.
- Das Signal löst den gleichnamigen Slot im AIController controller aus.

sig_results()

- Das Signal wird ausgelöst, wenn auf "Results" geclickt wird.
- Das Signal löst den gleichnamigen Slot im controller aus.

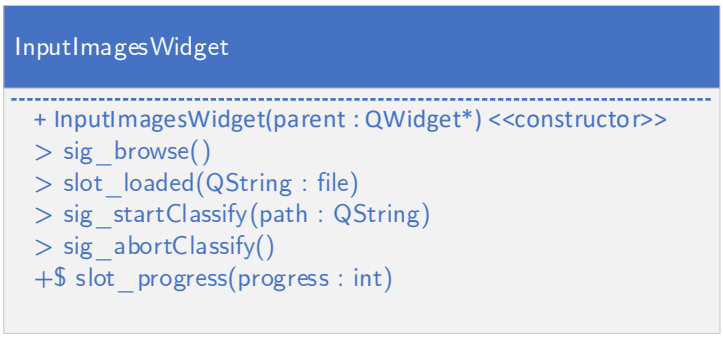
sig_showAugmentationPreview()

- Das Signal wird ausgelöst, wenn auf "Show Preview" geclickt wird.
- Das Signal löst den gleichnamigen Slot im DataAugmentationController aus.

slot_progress(progress : int)

- Dieser Slot wird von dem Klassifikationsplugin aufgerufen, wenn ein Fortschritt beim Training vermeldet wird.
- Der Slot ist mit dem gleichnamigen Signal in ClassifierTrainer verbunden.

3.2.6 InputImagesWidget



Klassenbeschreibung

〉 Das `InputImagesWidget`, bietet die Möglichkeit Eingabebilder für eine Klassifizierung anzugeben, diese werden dann auch angezeigt.

Methoden

sig_browse()

- Das Signal wird ausgelöst, wenn auf das "Ordner" Icon geclickt wird.
- Das Signal wird mit QFileDialog verbunden.

sig_startClassify()

- Das Signal wird ausgelöst, wenn auf "Classify" geclickt wird.
- Das Signal löst den gleichnamigen Slot im AIController aus

sig_abortClassify()

- Das Signal wird ausgelöst, wenn auf "Cancel" geclickt wird.
- Das Signal löst den gleichnamigen Slot im AIController aus

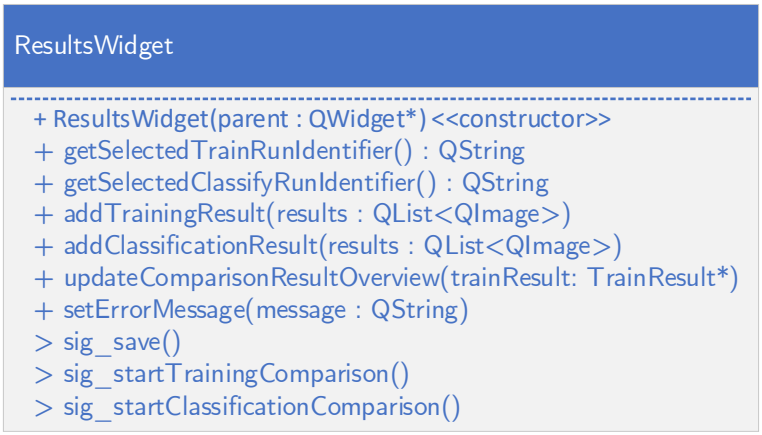
slot_loaded(file : QString)

- Dieser Slot wird von QFileDialog aufgerufen, falls dort ein Pfad übernommen wird.
- Der Slot ist mit dem gleichnamigen Signal in QFileDialog verbunden.

slot_progress(progress : int)

- Dieser Slot wird von dem Klassifikationsplugin aufgerufen, wenn bei der Klassifizierung der Eingabebilder Fortschritt gemeldet wird.
- Der Slot ist mit dem gleichnamigen Signal in ClassifierTrainer verbunden.

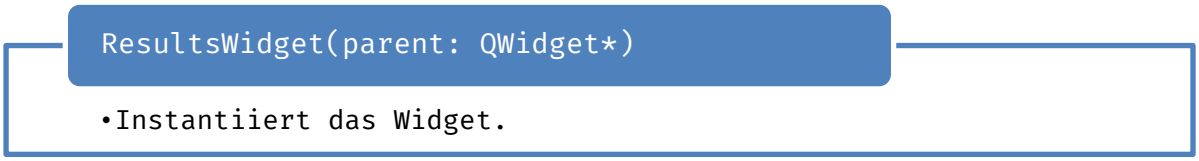
3.2.7 ResultsWidget



Klassenbeschreibung

〉 Das `ResultsWidget`, zeigt die Ergebnisse eines Trainings oder einer Klassifikation an. Es ermöglicht auch das Abspeichern dieser Ergebnisse.

Konstruktoren



Methoden

`getSelectedTrainRunIdentifier() : QString`

- Gibt einen eindeutigen Bezeichner zurück, um den zum Vergleich ausgewählten, gespeicherten Trainingsdurchlauf eindeutig zu identifizieren.

`getSelectedClassifyRunIdentifier() : QString`

- Gibt einen eindeutigen Bezeichner zurück, um den zum Vergleich ausgewählten, gespeicherten Klassifizierungsdurchlauf eindeutig zu identifizieren.

`addTrainingResult(results: QList<QImage>)`

- Setzt die anzuzeigenden Trainingsergebnisse.

`addClassificationResults(results: QList<QImage>)`

- Setzt die anzuzeigenden Klassifizierungsergebnisse.

`updateComparisonResultOverview(trainingResult : TrainingResult*)`

- Aktualisiert die Übersicht der Ergebnisse zum Vergleichen.

`setErrorMessage(message: QString)`

- Setzt die anzuzeigende Fehlermeldung

`sig_save()`

- Das Signal wird ausgelöst, wenn auf "Save" geklickt wird.
- Das Signal löst den gleichnahmigen Slot im ResultsController aus.

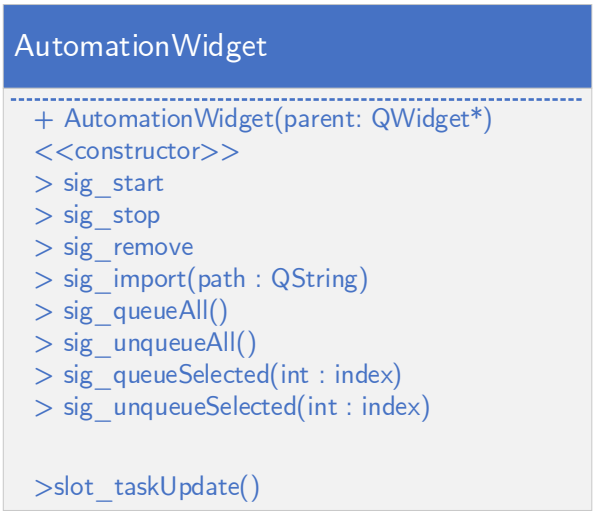
`sig_startTrainingComparison()`

- Gibt an, dass der Benutzer einen neuen Tab im Ergebnisfenster geöffnet hat, um Trainingsdurchläufe zu vergleichen.
- Das Signal löst den gleichnahmigen Slot im ResultsController aus.

`sig_startClassificationComparison()`

- Gibt an, dass der Benutzer einen neuen Tab im Ergebnisfenster geöffnet hat, um Trainingsdurchläufe zu vergleichen.
- Das Signal löst den gleichnahmigen Slot im ResultsController aus.

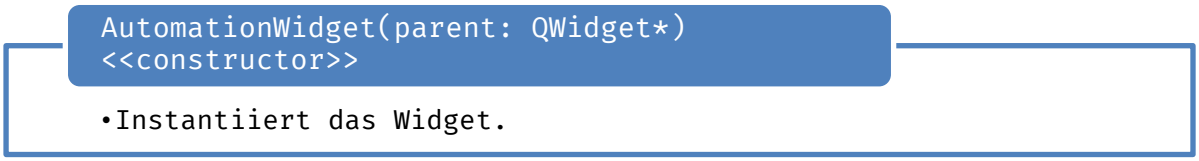
3.2.8 AutomationWidget



Klassenbeschreibung

〉 Dieses Widget stellt den Import von Tasks zur Batchverarbeitung dar und zeigt den Fortschritt von diesen.

Konstruktoren



Methoden

sig_start ()

- Wird ausgelöst, wenn der Start-Knopf gedrückt wird.
- Dieses Signal ist mit dem gleichnamigen Slot in AutomationController verbunden.

sig_stop()

- Wird durch drücken des Stop-Knopfs ausgelöst.
- Dieses Signal ist mit dem gleichnamigen Slot in AutomationController verbunden.

sig_remove()

- Wird durch drücken des Löschen-Knops ausgelöst.
- Dieses Signal ist mit dem gleichnamigen Slot in AutomationController verbunden.

sig_import(path : QString)

- Wird durch drücken des "Importiere neue Aufgaben"-Knopfs ausgelöst.
- Dieses Signal ist mit dem gleichnamigen Slot in AutomationController verbunden.

sig_queueAll()

- Wird durch drücken des ">>"-Knops ausgelöst.
- Dieses Signal ist mit dem gleichnamigen Slot in AutomationController verbunden.

sig_unqueueAll()

- Wird durch drücken des "<<"-Knopfs ausgelöst.
- Dieses Signal ist mit dem gleichnamigen Slot in AutomationController verbunden.

sig_queueSelected(int : index)

- Wird durch drücken des ">|"-Knopfs ausgelöst.
- Dieses Signal ist mit dem gleichnamigen Slot in AutomationController verbunden.

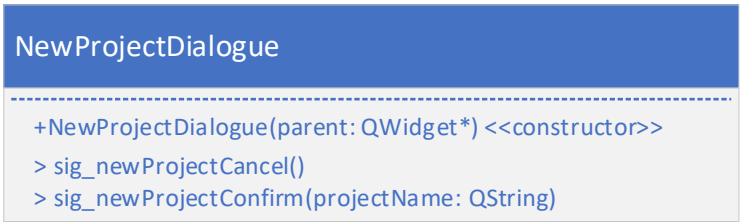
sig_unqueueSelected(int : index)

- Wird durch drücken des "|<"-Knopfs ausgelöst.
- Dieses Signal ist mit dem gleichnamigen Slot in AutomationController verbunden.

slot_taskUpdate()

- Wird verwendet um Änderungen der Tasks entgegenzunehmen.
- Dieser Slot ist mit dem gleichnamigen Signal in AutomationController verbunden.

3.2.9 NewProjectDialog



Klassenbeschreibung

〉 Dialogfenster, auf dem der Benutzer ein neues Projekt parametrisieren und anlegen kann.

Konstruktoren

NewProjectDialog(parent: QWidget*) <<constructor>>

- Instantiiert das Fenster.

Methoden

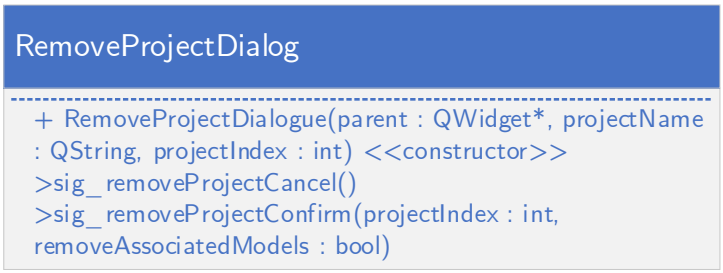
sig_newProjectCancel()

- Das Signal wird ausgelöst, wenn auf "Cancel" geclickt wird.
- Das Signal löst den gleichnamigen Slot im ProjectController aus.

sig_newProjectConfirm(projectName: QString)

- Das Signal wird ausgelöst, wenn auf "OK" geklickt wird.
- Das Signal löst den gleichnamigen Slot im ProjectController aus

3.2.10 RemoveProjectDialog



Klassenbeschreibung

〉 Dialogfenster, auf dem der Benutzer die Löschung eines Projekts spezifizieren und bestätigen kann.

Konstruktoren

RemoveProjectDialog(parent: QWidget*) <<constructor>>

- Instantiiert das Fenster.

Methoden

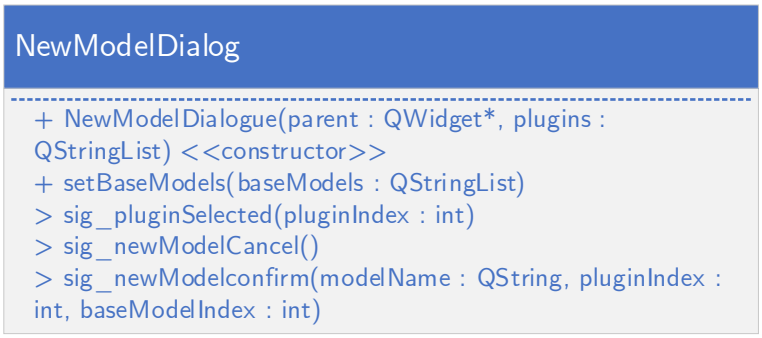
sig_removeProjectCancel()

- Schließt das Fenster.

sig_removeProjectConfirm(projectIndex : int, removeAssociatedModels : bool)

- Entfernt das ausgewählte Projekt. Wenn true übermittel wird werden alle Modelle auf dem Projekt gelöscht-

3.2.11 NewModelDialog



Klassenbeschreibung

〉 Dialogfenster, auf dem der Benutzer ein neues Model parametrisieren und anlegen kann.

Konstruktoren

NewModelDialog(parent: QWidget*, plugins : QStringList) <<constructor>>

- Instantiiert das Fenster.

Methoden

setBaseModels(baseModels: QStringList)

- Setzt die anzuzeigenden Basismodelle.

sig_pluginSelected(pluginIndex: int)

- Das Signal wird ausgelöst, wenn auf ein Plugin in der ComboBox gelclickt wird.
- Das Signal löst den gleichnamigen Slot im Modelcontroller aus.

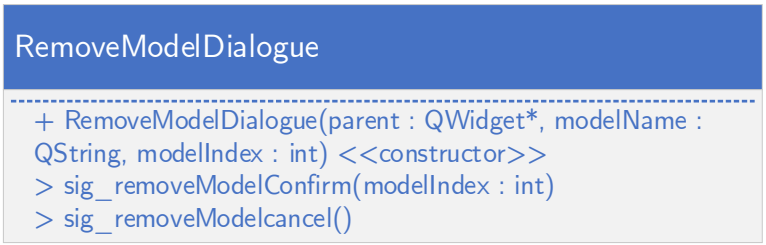
sig_newModelCancel()

- Das Signal wird ausgelöst, wenn auf "Cancel" geclickt wird.
- Das Signal löst den gleichnamigen Slot im Modelcontroller aus.

sig_newModelConfirm(modelName: QString, pluginIndex: int, baseModelIndex: int)

- Das Signal wird ausgelöst, wenn auf "OK" geclickt wird.
- Das Signal löst den gleichnamigen Slot im Modelcontroller aus.

3.2.12 RemoveModelDialog



Klassenbeschreibung

〉 Dialogfenster auf dem der Benutzer die Löschung eines Modells bestätigen kann.

Konstruktoren

RemoveModelDialog (parent: QWidget*, modelName: QString, modelIndex: int) <<constructor>>

- Instantiiert das Fenster.

Methoden

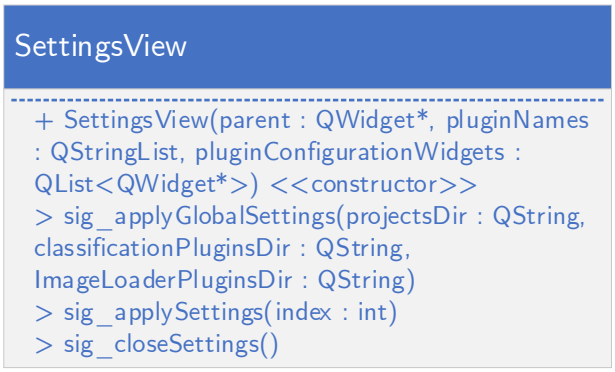
sig_removeModelConfirm(modelIndex: int)

- Das Signal wird ausgelöst, wenn auf "OK" geclickt wird.
- Das Signal löst den gleichnamigen Slot im ProjectController aus.

sig_removeModelCancel()

- Das Signal wird ausgelöst, wenn auf "Cancel" geclickt wird.
- Das Signal löst den gleichnamigen Slot im ProjectController aus

3.2.13 SettingsDialog



Klassenbeschreibung

〉 Dialogfenster auf dem globale sowie auch pluginspezifische Einstellungen angeboten werden

Konstruktoren

```
SettingsDialog (parent : QWidget*, pluginNames :
QStringList, pluginConfigurationWidgets :
QList<QWidget*>)
```

•Instantiiert das Fenster.

Methoden

```
sig_applyGlobalSettings(projectsDir : QString,
classificationPluginsDir : QString,
ImageLoaderPluginsDir : QString)
```

•Das Signal wird ausgelöst, wenn auf "apply" geclickt wird und der Benutzer sich bei den Pluginspezifischen Einstellungen befindet.
•Das Signal löst den gleichnamigen Slot im SettingsController aus.

```
sig_applySettings(index : int)
```

•Das Signal wird ausgelöst, wenn auf "apply" geclickt wird und der Benutzer sich bei den Globaleneinstellungen befindet.
•Das Signal löst den gleichnamigen Slot im SettingsController aus

```
sig_closeSettings()
```

•Das Signal wird ausgelöst, wenn auf "Close" geclickt wird.
•Das Signal löst den gleichnamigen Slot im SettingsController aus

4 Controller

Die *Controllerschicht* dient in MVC der Vermittlung zwischen *View* und *Model*.

4.1 Entwurf

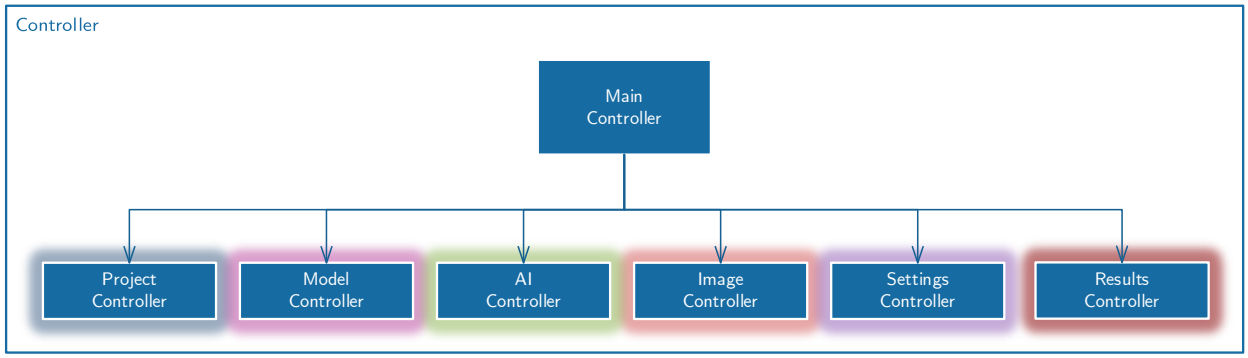


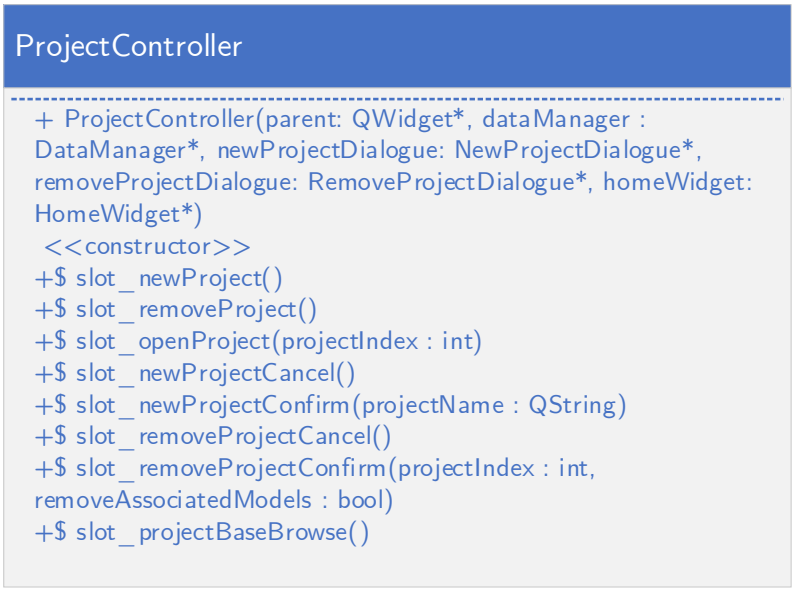
Abbildung 6 : Übersicht des Controllers

Das Controller Paket ist dafür verantwortlich, Benutzereingaben, die in der graphischen Benutzer Oberfläche verfasst wurden, entgegenzunehmen, und diese jeweils an die richtige Stelle im Modell weiterzuleiten. Entsprechend wird auch die GUI durch den Controller auf Änderungen oder Erneuerungen in dem Model, wie z.B. Ergebnisse oder Fortschritte, aufmerksam gemacht. Hierfür gibt es neben dem Hauptcontroller, der das `MainWindow` kontrolliert, für jeden eigenständigen Teil des Programms einen separaten Controller:

ProjectController	Dieser Controller verwaltet Widgets und Dialoge für die Auswahl und Änderung der Projekte und kommuniziert dabei mit dem `DataManager`.
ModelController	Dieser Controller verwaltet Widgets und Dialoge für die Auswahl und Änderung der Klassifikationsmodelle und kommuniziert dabei mit dem `ModelManager`.
AIController	Dieser Controller verwaltet unter anderem das `AITrainingTab` und kommuniziert dabei mit dem `ClassifierTrainer`.
ImageController	Dieser Controller verwaltet das `ImageInspectionWidget` und `ImportFilesWidget`. Dabei wird mit dem `ImageSearcher` kommuniziert.
SettingsController	Dieser Controller verwaltet das `SettingsWidget` und kommuniziert dabei mit dem `SettingsManager`.
ResultsController	Dieser Controller verwaltet das `ResultsWidget`.

4.2 Klassen

4.2.1 ProjectController



Klassenbeschreibung

〉 Die Klasse bietet Slots für das `HomeWidget`, sowohl auch für den `NewProjectDialog` und `RemoveProjectDialog`. Die Klasse verwaltet Aktionen bezüglich des Anlegens und Entfernens von Projekten sowie deren Verzeichnissen.

Attribute

〉 Referenz zum `DataManager`

〉 Referenz zum `NewProjectDialog`

〉 Referenz zum `RemoveProjectDialog`

〉 Referenz zum `HomeWidget`

Konstruktoren

```
ProjectController(parent: QWidget*, dataManager :
DataManager*, newProjectDialogue: NewProjectDialogue*,
removeProjectDialogue: RemoveProjectDialogue*,
homeWidget: HomeWidget*)
```

- Instantiiert den Controller und übergibt ihm die Referenzen

Methoden

`slot_newProject ()`

- Dieser Slot wird vom HomeWidget aufgerufen, wenn der NewProjectDialog angezeigt werden soll.
- Der Slot ist mit dem gleichnamigen Signal im HomeWidget verbunden.

`slot_removeProject ()`

- Dieser Slot wird vom HomeWidget aufgerufen, wenn der RemoveProjectDialog angezeigt werden soll.
- Der Slot ist mit dem gleichnamigen Signal im HomeWidget verbunden.

`slot_openProject(projectIndex: int)`

- Dieser Slot wird vom HomeWidget aufgerufen, wenn das ausgewählte project geladen werden soll.
- Der Slot ist mit dem gleichnamigen Signal im HomeWidget verbunden.

`slot_newProjectCancel()`

- Dieser Slot wird vom NewProjectDialog aufgerufen, wenn kein neues Projekt angelegt werden soll. Der Dialog wird darauf geschlossen.
- Der Slot ist mit dem gleichnamigen Signal im NewProjectDialog verbunden.

`slot_newProjectConfirm(projectName : QString)`

- Dieser Slot wird vom NewProjectDialgoue aufgerufen, wenn ein neues Projekt angelegt werden soll, mit den Eingaben die im Dialog definiert wurden. Der Dialog wird darauf geschlossen.
- Der Slot ist mit dem gleichnamigen Signal in NewProjectDialog verbunden.

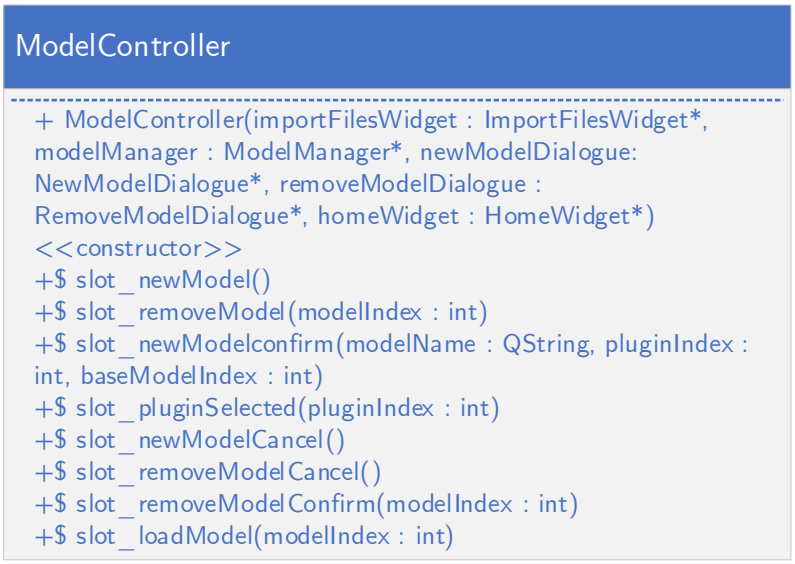
`slot_removeProjectCancel()`

- Dieser Slot wird vom RemoveProjectDialog aufgerufen, wenn das ausgewählte Projekt nicht gelöscht werden soll. Der Dialog wird darauf geschlossen.
- Der Slot ist mit dem gleichnamigen Signal in RemoveProjectDialog verbunden.

`slot_removeProjectConfirm(projectIndex : index removeAssociatedModells : bool)`

- Dieser Slot wird vom RemoveProjectDialog aufgerufen, wen das ausgewählte Projekt gelöscht werden soll. Der Dialog wird darauf geschlossen.
- Der Slot ist mit dem gleichnamigen Signal in RemoveProjectDialog verbunden.

4.2.2 ModelController



Klassenbeschreibung

〉 Die Klasse bietet Slots für das `ImportFilesWidget`, sowohl auch für den `NewModelDialog` und `RemoveModelDialog`. Die Klasse verwaltet Aktionen bezüglich des Anlegens und Entfernens von Modellen.

Attribute

〉 Referenz zum `DataManager`

〉 Referenz zum `NewModelDialog`

〉 Referenz zum `RemoveModelDialog`

〉 Referenz zum `ImportFilesWidget`

Konstruktoren

```
ModelController(parent: QWidget*, importFilesWidget :
ImportFilesWidget*, dataManager : DataManager*,
newModelDialogue: NewModelDialogue*,
removeModelDialogue: RemoveModelDialogue*)
```

•Instantiiert den Controller und übergibt ihm die Referenzen

Methoden

`slot_newModel ()`

- Dieser Slot wird vom ImportFilesWidget aufgerufen, wenn der NewModelDialog angezeigt werden soll.
- Der Slot ist mit dem gleichnamigen Signal im ImportFilesWidget verbunden.

`slot_removeModel(modelIndex: int)`

- Dieser Slot wird vom ImportFilesWidget aufgerufen, wenn der RemoveModelDialog angezeigt werden soll.
- Der Slot ist mit dem gleichnamigen Signal im HomeWidget verbunden.

`slot_newModelCancel()`

- Dieser Slot wird vom NewProjectDialog aufgerufen, wenn kein neues Modell angelegt werden soll. Der Dialog wird darauf geschlossen.
- Der Slot ist mit dem gleichnamigen Signal im ImportFilesWidget verbunden.

`slot_newModelConfirm(modelName: QString, pluginIndex: int, baseModelIndex: int)`

- Dieser Slot wird vom NewModelDialgoue aufgerufen, wenn ein neues Modell angelegt werden soll, mit den Eingaben die im Dialog definiert wurden. Der Dialog wird darauf geschlossen.
- Der Slot ist mit dem gleichnamigen Signal in NewModelDialog verbunden.

`slot_removeModelConfirm(modelIndex: int)`

- Dieser Slot wird vom RemoveProjectDialgoue aufgerufen, wenn das ausgewählte Model gelöscht werden soll. Der Dialog wird darauf geschlossen.
- Der Slot ist mit dem gleichnamigen Signal in RemoveModelDialog verbunden.

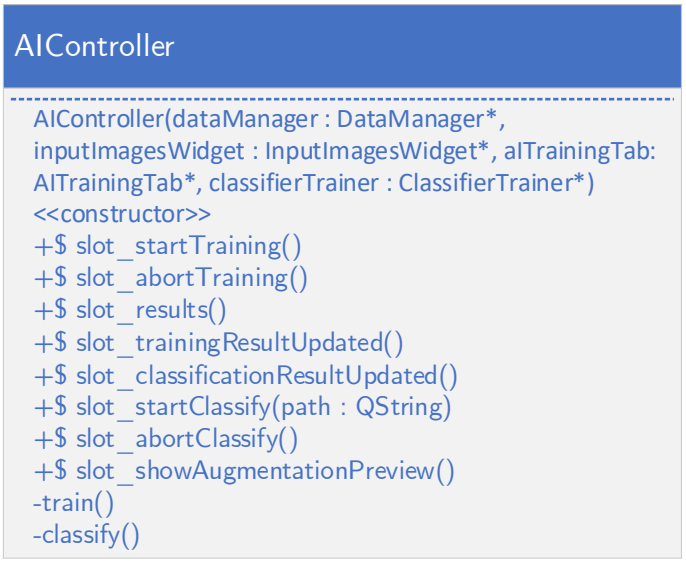
`slot_loadModel(modelIndex: int)`

- Dieser Slot wird vom ImportFilesWidget aufgerufen, wenn das ausgewählte Model geladen werde soll.
- Der Slot ist mit dem gleichnamigen Signal in RemoveModelDialog verbunden.

`slot_pluginSelected(pluginIndex: int)`

- Dieser Slot wird vom NewModelDialog aufgerufen, wenn in der Plugin ComboBox ein Plugin geclickt wird.
- Der Slot ist mit dem gleichnamigen Signal in RemoveModelDialog verbunden.

4.2.3 AIController



Klassenbeschreibung

〉 Die Klasse bietet Slots für das `AITrainingWidget`, `InputWidget` und `AlgorithmDialog`. Die Klasse verwaltet Aktionen bezüglich des Startens und Abbrechen von Trainings- und Klassifizierungsprozessen, sowohl auch der Weiterleitung von dessen Fortschritten.

Attribute

〉 Referenz zum `DataManager`

〉 Referenz zum `ClassifierTrainer`

〉 Referenz zum `AITrainingWidget`

〉 Referenz zum `InputImagesWidget`

Konstruktoren

```
AIController(dataManager : DataManager*,
classifierTrainer: classifierTrainer*,
aiTrainingWidget : AITrainingWidget*,
inputImageWidget : InputImagesWidget*)
```

•Instantiiert den Controller und übergibt ihm die Referenzen

Methoden

slot_startTraining()

- Dieser Slot wird vom AITrainingWidget aufgerufen, wenn der Trainingsprozess initiert werden soll.
- Der Slot ist mit dem gleichnamigen Signal im AITrainingWidget verbunden.

slot_abortTraining()

- Dieser Slot wird vom AITrainingWidget aufgerufen, wenn ein angefangens Training abgerochen werden soll.
- Der Slot ist mit dem gleichnamigen Signal im AITrainingWidget verbunden.

slot_results()

- Dieser Slot wird vom AITrainingWidget aufgerufen, wenn das Results widget angezeigt werden soll.
- Der Slot ist mit dem gleichnamigen Signal im AITraining verbunden.

slot_startClassify(path : String)

- Dieser Slot wird vom InputImagesWidget aufgerufen, wenn der Klassifizierungsprozess initiert werden soll.
- Der Slot ist mit dem gleichnamigen Signal in InputImagesWidget verbunden.

slot_abortClassify(path : String)

- Dieser Slot wird vom InputImagesWidget aufgerufen, wenn ein angefangener Klassifizierungsprozess abgerochen werden soll.
- Der Slot ist mit dem gleichnamigen Signal in InputImagesWidget verbunden.

slot_trainingResultUpdated()

- Dieser Slot wird vom ClassifierTrainer aufgerufen, wenn die Ergebnisse eines Trainingsprozesses bei ihm anliegen.
- Der Slot ist mit dem gleichnamigen Signal in ClassifierTrainer verbunden.

slot_classificationResultUpdated()

- Dieser Slot wird vom ClassifierTrainer aufgerufen, wenn die Ergebnisse eines Klassifizierungsprozesses bei ihm anliegen.
- Der Slot ist mit dem gleichnamigen Signal in ClassifierTrainer verbunden.

slot_showAugmentationPreview()

- Veranlasst die Anzeige der Augmentationvorschau durch das Plugin des verwendeten Modells

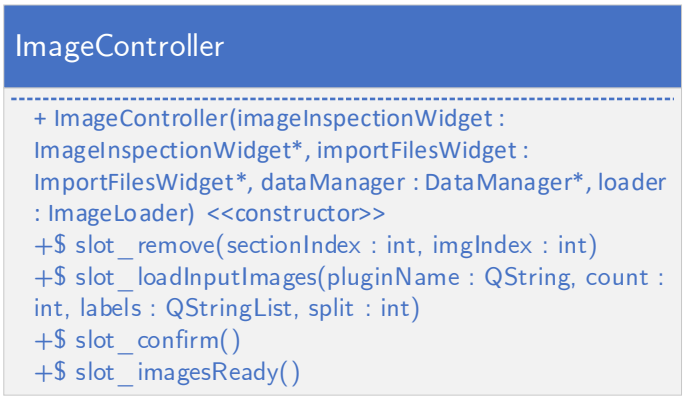
train()

- Fordert den TrainerClassifier auf, anhand Benutzerdefinierter Daten, einen Trainingsprozess anzulegen und auszuführen.

classify()

- Fordert den TrainerClassifier auf, anhand Benutzerdefinierter Daten, einen Klassifizierungsprozess anzulegen und auszuführen.

4.2.4 ImageController



Attribute

- Referenz zum `DataManager`
- Referenz zum `ImageSearcher`
- Referenz zum `ImportFilesWidget`
- Referenz zum `ImageInspectionWidget`

Klassenbeschreibung

- Die Klasse bietet Slots für das `ImportFilesWidget`. Die Klasse verwaltet Aktionen bezüglich des Parametrisierens und Starten von Ladeprozessen für Trainingsbilder.

Konstruktoren

```
ImageController(imageInspectionWidget :
ImageInspectionWidget*, importFilesWidget :
ImportFilesWidget*, dataManager : DataManager*, loader :
ImageLoader)
```

•Instantiiert den Controller und übergibt ihm die Referenzen

Methoden

`slot_remove(sectionIndex : int, imgIndex : int)`

- Dieser Slot wird vom ImageInspectionWidget aufgerufen, wenn die ausgewählten bilder zu entfernen sind.
- Der Slot ist mit dem gleichnamigen Signal im ImageInspectionWidget verbunden.

`slot_loadInputImages(pluginName : QString, count : int, labels : QStringList, split : int)`

- Dieser Slot wird vom ImportFilesWidget aufgerufen, wenn der Such- und Ladeprozess der Trainingsbilder initiert werden soll.
- Der Slot ist mit dem gleichnamigen Signal im ImportFilesWidget verbunden.

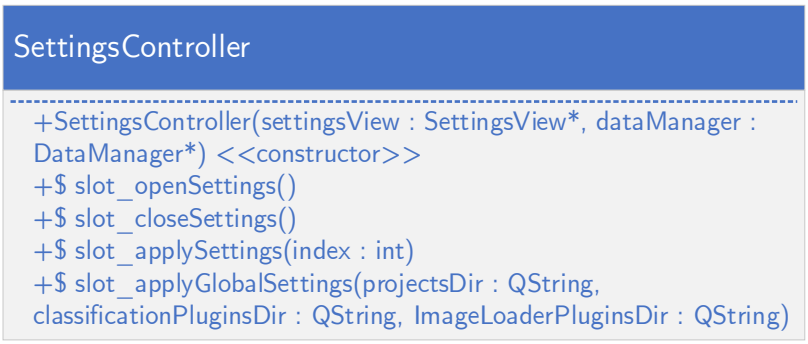
`slot_confirm()`

- Dieser Slot wird vom ImageInspectionWidget aufgerufen, wenn die heruntergeladen Bilder zu dem Datensatz hinzugefüt werden sollen.
- Der Slot ist mit dem gleichnamigen Signal im ImageInspectionWidget verbunden.

`slot_imagesReady()`

- Dieser Slot wird vom ImageSearcher aufgerufen, wenn die Bilder einer Suchanfrage bereit zum anzeigen sind.
- Der Slot ist mit dem gleichnamigen Signal im ImageSearcher verbunden.

4.2.5 SettingsController



Klassenbeschreibung

〉 Die Klasse bietet Slots für das `MainWindow` und den `SettingsDialog`. Die Klasse verwaltet Aktionen bezüglich des Einsehens und Ändern von Plugin spezifischen und Globalen Einstellungen.

Attribute

〉 Referenz zum `SettingsDialog`

〉 Referenz zum `DataManager`

Konstruktoren

```
SettingsController(settingsView : SettingsView*,
dataManager : DataManager*)
```

•Instantiiert den Controller und übergibt ihm die Referenzen

Methoden

`slot_opensettings()`

- Dieser Slot wird vom MainWindow aufgerufen, wenn der SettingsDialog angezeigt werden soll.
- Der Slot ist mit dem gleichnamigen Signal im MainWindow verbunden.

`slot_closeSettings()`

- Dieser Slot wird vom SettingsDialog aufgerufen, wenn der SettingsDialog geschlossen werden soll.
- Der Slot ist mit dem gleichnamigen Signal im SettingsDialog verbunden.

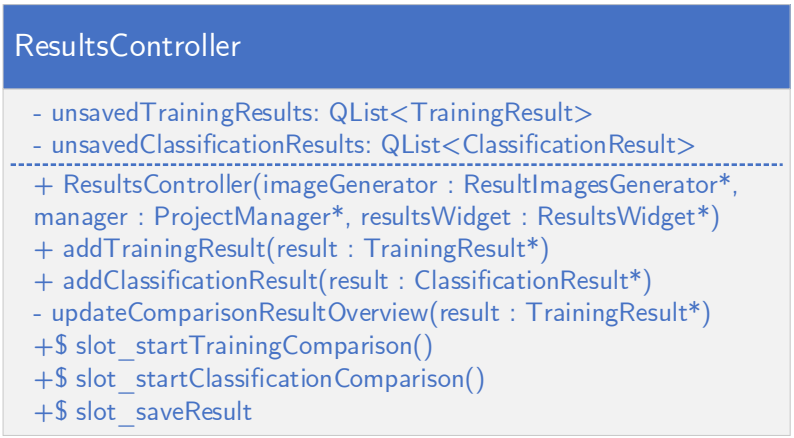
`slot_settingsApply(index: int)`

- Dieser Slot wird vom SettingsDialog aufgerufen, wenn die Benutzereingaben in den Einstellungen auf der indizierten Seite übernommen werden sollen.
- Der Slot ist mit dem gleichnamigen Signal im SettingsDialog verbunden.

`slot_applyGlobalSettings(projectsDir : QString, classificationPluginsDir : QString, PictureLoaderPluginsDir : QString)`

- Dieser Slot wird vom SettingsDialog aufgerufen, wenn die Benutzereingaben in den Gloableneinstellungen übernommen werden sollen.
- Der Slot ist mit dem gleichnamigen Signal im SettingsDialog verbunden

4.2.6 ResultsController



Klassenbeschreibung

› Der `ResultsController` dient zur Übermittlung von Trainings- und Klassifizierungsergebnissen an das `ResultsWidget`.

Attribute

› Referenz zum `ResultsWidget`

› Referenz zum `ProjectManager`

Konstruktoren

```
ResultsController(imageGenerator :
ResultImagesGenerator*,  projectManager :
ProjectManager*, resultsWidget : ResultsWidget*)
```

•Instantiiert den ResultsController und übergibt ihm die benötigten Referenzen

Methoden

`addTrainingResult(result : TrainingResult*)`

- Diese Methode holt sich vom ResultsWidget den Bezeichner des zu vergleichenden Trainingsdurchlaufes und liefert die vom ProjectManager geholten Trainingsergebnisse an das ResultWidget weiter, sodass der Benutzer sie vergleichen kann.

`addClassificationResult(result : ClassificationResult*)`

- Diese Methode holt sich vom ResultsWidget den Bezeichner des zu vergleichenden Klassifizierungsdurchlaufes und liefert die vom ProjectManager geholten Trainingsergebnisse an das ResultWidget weiter, sodass der Benutzer sie vergleichen kann.

`updateComparisonResultOverview(result : TrainingResult*)`

- Bietet eine Übersicht über zu vergleichende Trainingsergebnisse an.

`slot_startTrainingComparison()`

- Dieser Slot wird vom ResultsWidget aufgerufen, wenn ein neuer Trainingsdurchlauf zum Vergleichen geöffnet werden soll.
- Dieser Slot ist mit dem gleichnahmigen Signal im ResultsWidget verbunden.

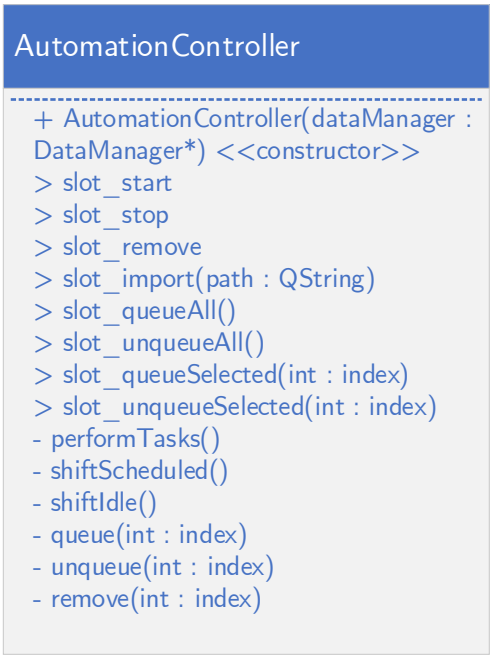
`slot_startTrainingComparison()`

- Dieser Slot wird vom ResultsWidget aufgerufen, wenn ein neuer Klassifizierungsdurchlauf zum Vergleichen geöffnet werden soll.
- Dieser Slot ist mit dem gleichnahmigen Signal im ResultsWidget verbunden.

`slot_saveResult()`

- Dieser Slot wird vom ResultsWidget aufgerufen, wenn Trainings- oder Klassifizierungsergebnisse abgespeichert werden sollen.
- Dieser Slot ist mit dem gleichnahmigen Signal im ResultsWidget verbunden.

4.2.7 AutomationController



Klassenbeschreibung

〉 Der `AutomationController` ist verantwortlich für die Übermittlung des Fortschritts, sowie der aktuellen Tasks des `Automators`, an das `AutomationWidget`.

Attribute

〉 Referenz zum DataManager

Konstruktoren

`AutomationController(dataManager: DataManager*)`

- `Instantiiert den AutomationController und übergibt ihm die Referenz auf den DataManager.`

Methoden

slot_start()

- Dieser Slot wird von AutomationWidget aufgerufen, wenn die Batchverarbeitung gestartet werden soll.
- Dieser Slot ist mit dem gleichnamigen Signal in AutomationWidget verbunden.

>slot_stop()

- Dieser Slot wird von AutomationWidget aufgerufen, wenn die Batchverarbeitung gestoppt werden soll.
- Dieser Slot ist mit dem gleichnamigen Signal in AutomationWidget verbunden.

slot_remove()

- Dieser Slot wird von AutomationWidget aufgerufen, wenn ein Task entfernt werden soll.
- Dieser Slot ist mit dem gleichnamigen Signal in AutomationWidget verbunden.

slot_import(path : QString)

- Dieser Slot wird von AutomationWidget aufgerufen, wenn Tasks aus einer Datei importiert werden sollen.
- Dieser Slot ist mit dem gleichnamigen Signal in AutomationWidget verbunden.

slot_queueAll()

- Dieser Slot wird von AutomationWidget aufgerufen, wenn alle Tasks in Idle gescheduеld werden sollen.
- Dieser Slot ist mit dem gleichnamigen Signal in AutomationWidget verbunden.

slot_unqueueAll()

- Dieser Slot wird von AutomationWidget aufgerufen, wenn alle Schedueled Tasks in Idle verschoben weden sollen.
- Dieser Slot ist mit dem gleichnamigen Signal in AutomationWidget verbunden.

slot_queueSelected(int : index)

- Dieser Slot wird von AutomationWidget aufgerufen, wenn gewählte Tasks in Idle gescheduеld werden sollen.
- Dieser Slot ist mit dem gleichnamigen Signal in AutomationWidget verbunden.

slot_unqueueSelected(int : index)

- Dieser Slot wird von AutomationWidget aufgerufen, wenn gewählte Schedueled Tasks in Idle verschoben werden sollen.
- Dieser Slot ist mit dem gleichnamigen Signal in AutomationWidget verbunden.

4.2.8 Controller

Klassenbeschreibung

› Der `Controller` ist der Einstiegspunkt in das Programm und verwaltet alle anderen Controller.

Attribute

› Referenz zum `ModelController`

› Referenz zum `ProjectController`

› Referenz zum `AIController`

› Referenz zum `ImageController`

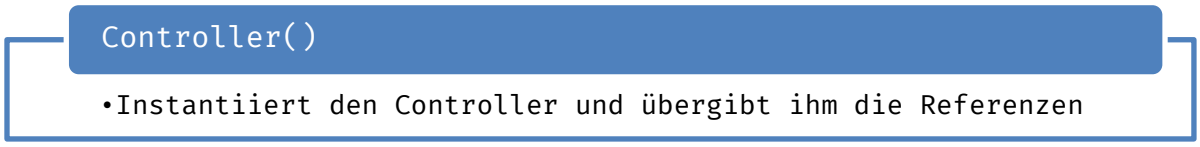
› Referenz zum `SettingsController`

› Referenz zum `ResultsController`

› Referenz zum `AutomationController`

› Referenz zum `HomeWidget`

Konstruktoren



5 Plugin

In diesem Kapitel werden die Klassen des Plugin-Frameworks der Anwendung näher erläutert. Man unterscheidet dabei zwischen Klassifizierungsplugins, Bildsammlerplugins und gemeinsamer Funktionalität, die Plugins unabhängig von ihrer konkreten Verwendung bereitstellen müssen.

5.1 Entwurf

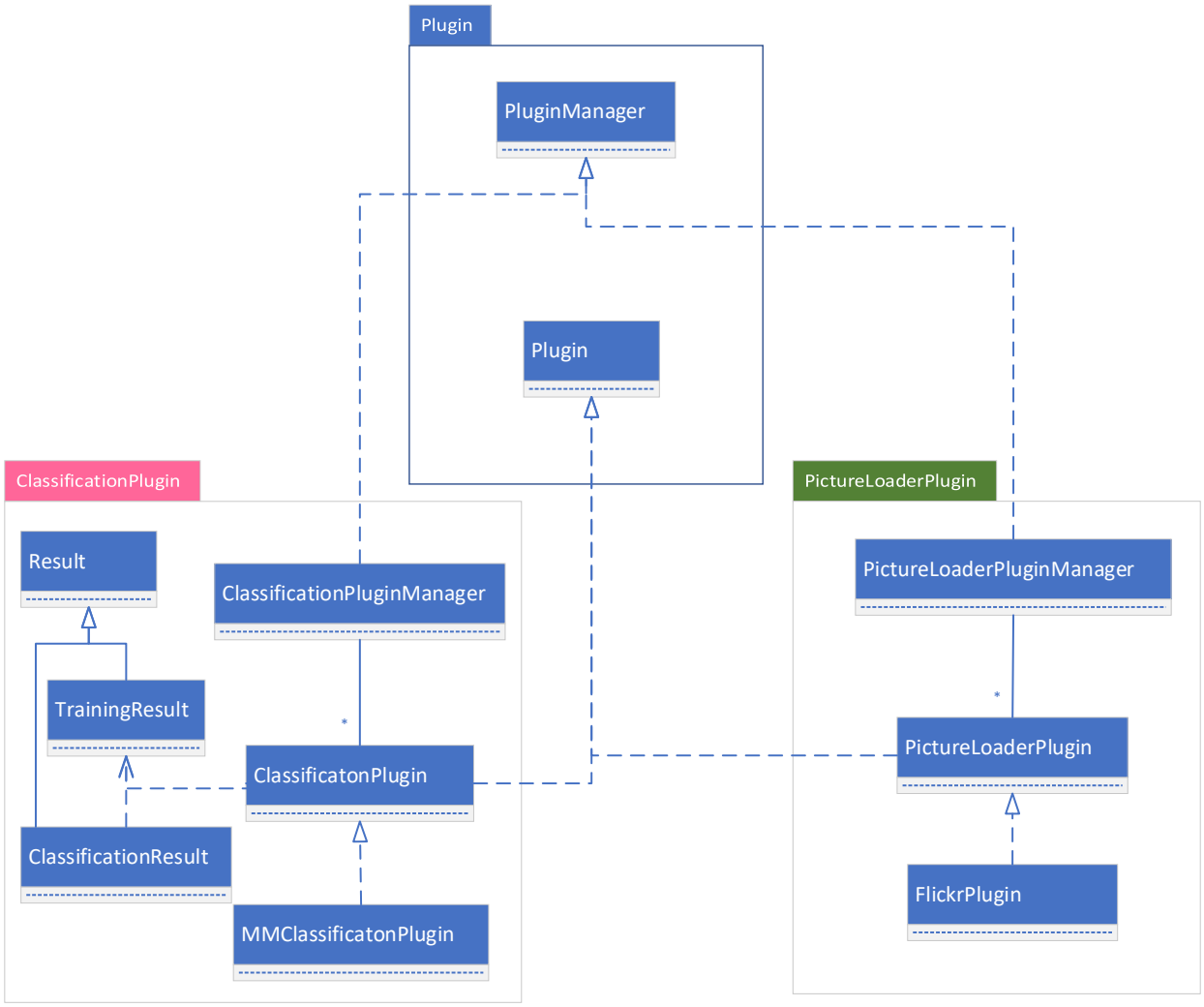


Abbildung 5: Plug-in Architektur

Die Erweiterbarkeit ist eine Kernkomponente unserer Anwendung. Um die Verwendung unterschiedlicher Klassifizierung -und Bildsammlerschnittstellen so einfach wie möglich zu machen, bedienen wir uns der Plugin Funktionalität von Qt. Da ein Qt Plugin in Form einer Dynamic-Linked-Library gespeichert wird, hat man durch dieses Vorgehen einige Vorteile gegenüber der Verwendung von Shared-Librarys durch `QLibrary`.

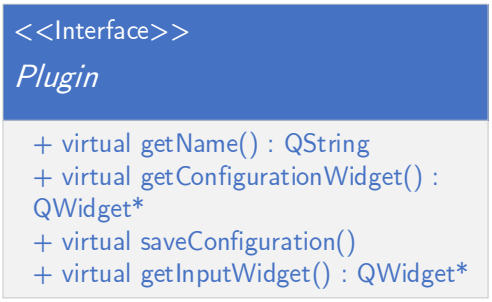
Beispielsweise wird überprüft, ob ein Plugin mit derselben Version von Qt wie die Anwendung verknüpft ist. Die Dynamic-Linked-Library ermöglicht es den Programmcode zur Laufzeit zu laden, sodass das Rahmenprogramm nach Hinzufügen eines neuen Plugins nicht neu kompiliert werden muss.

Unsere Anwendung verwaltet Klassifizierung- und Bildsammlerplugins durch die Klassen `ClassificationPluginManager` und `PictureLoaderPluginManager`. Sie sind ebenfalls die Einstiegspunkte in das Plugin-Framework, damit das Rahmenprogramm auf die Funktionalität der Plugins zugreifen kann. Sofern man mit der Software weitere Arten von Plugins verwaltet will, muss lediglich die Schnittstelle `PluginManager` implementiert werden. Will man dagegen ein neues Klassifizierungsplugin bzw. Bildsammlerplugin hinzufügen, so muss die jeweilige Schnittstelle `ClassificationPlugin` bzw. `PictureLoaderPlugin` implementiert werden und als .dll Datei dem entsprechenden Pluginordner hinzugefügt werden. Das Klassifizierungsplugin hat je nach auszuführender Aktion (Training oder Klassifizierung) die Verantwortung entsprechende Ergebnisse in Form der Klassen `TrainingResult` und `ClassificationResult` an den `ClassificationPluginManager` zurückzugeben. Diese Ergebnisse können dann an die Rahmenanwendung weitergegeben werden, um sie in ein maschinenlesbares Format zu bringen. Die Bildsammlerplugins dagegen bekommen von der Rahmenanwendung über den `PictureLoaderPluginManager` einen Ordner zum Hineinladen der Bilder zur Verfügung gestellt.

Im Folgenden wird auf die Klassen der Plug-in-Architektur und der damit verbundenen Funktionalität näher eingegangen.

5.2 Klassen

5.2.1 Plugin



Klassenbeschreibung

〉 Diese Schnittstelle gibt grundlegende Methoden für ein Plugins dieser Anwendung vor.

Methoden

virtual getName(): QString

- Gibt den Anzeigenamen des Plugins zurück.

virtual getConfigurationWidget(): QWidget*

- Gibt ein Widget mit den pluginspezifischen Einstellungen zurück.

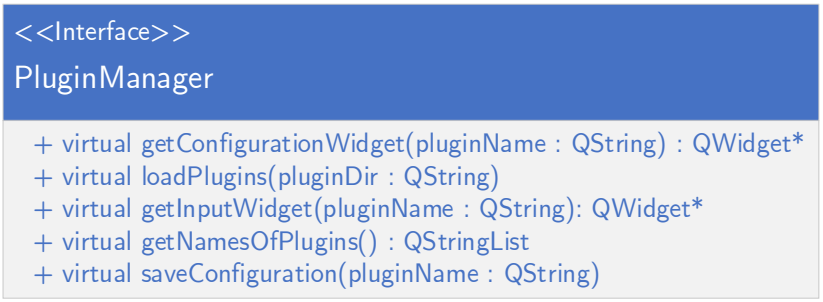
virtual saveConfiguration()

- Speichert die verwendeten Einstellungen des Plugins.

virtual getInputWidget(): QWidget*

- Gibt ein Widget mit Eingabefeldern zur Konfiguration eines Trainings mit diesem Plugin zurück.

5.2.2 PluginManager



Klassenbeschreibung

〉 Diese Schnittstelle beinhaltet grundlegende Methoden, die die Klassen `ClassificationPluginManager` und `PictureLoaderPluginManager` zur Verwaltung des Plugins bereitstellen müssen.

Methoden

`virtual getConfigurationWidget(pluginName: QString): QWidget*`

- Gibt das Konfigurationswidget des mit pluginName indizierten Plugins zurück.

`virtual getInputWidget(pluginName: QString): QWidget*`

- Gibt ein Widget mit Eingabefeldern zur Konfiguration eines Trainings mit diesem Plugin zurück.

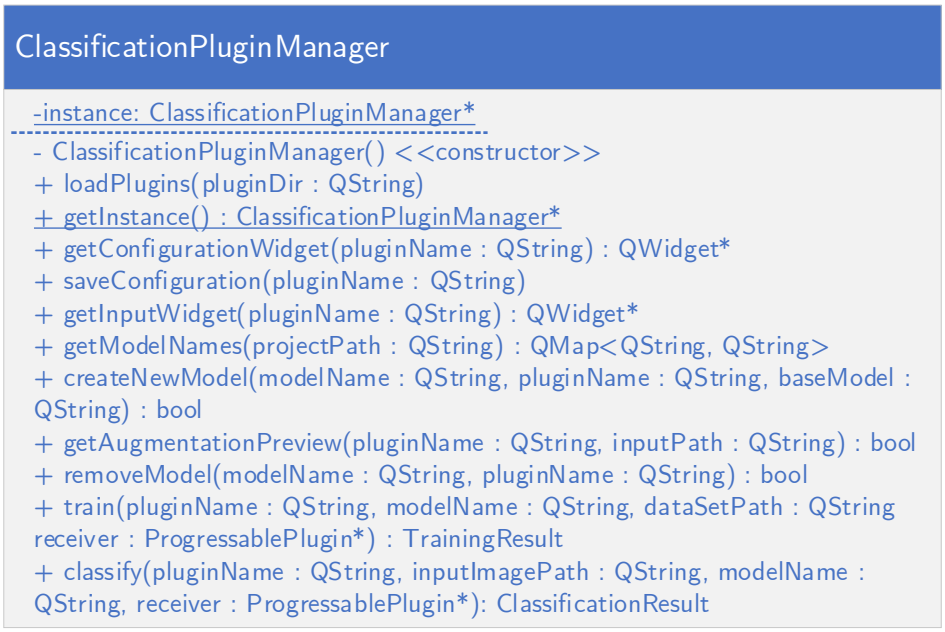
`virtual getNamesOfPlugins(): QStringList`

- Gibt eine Liste mit den Namen aller schon geladenen Plugins zurück.

`virtual saveConfiguration(pluginName: QString)`

- Speichert die zu verwendenen Einstellungen des Plugins.

5.2.3 ClassificationPluginManager



Klassenbeschreibung

- Anwendungszentrale Verwaltung der Klassifizierungsplugins.
- Die Klasse implementiert die Interface `PluginManager`.
- Die im Interface beschriebenen virtuellen Funktionen werden nicht mehr näher beschrieben.
- Der Einsatz des *Singleton*-Entwurfsmuster stellt sicher, dass nur eine Instanz vom `ClassificationPluginManager` existiert.

Attribute

- Statische *Singleton*-Instanz
- Referenz zum `ClassificationPlugin`

Konstruktoren

`ClassificationPluginManager()`

- `Privater Konstruktor des Singleton-Objektes`

Methoden

`loadPlugins(pluginDir : QString)`

- Lädt die Plugins am angegebenen Pfad.

`getInstance(): ClassificationPluginManager`

- Gibt eine Referenz zur der einzigen Instanz von ClassificationPluginManager zurück und legt diese Instanz an, falls noch keine existiert.

`getModelNames(projectPath: QString): QMap<QString, QString>`

- Gibt eine Liste mit den Namen aller bereits gespeicherten, trainierten Modelle zurück.

`createNewModel(modelName: QString,pluginName: QString, baseModel: QString): bool`

- Legt neues Modell mit den angegebenen Namen des Modells, des Plugins und des Basismodells an. Gibt false zurück, falls das Verfahren nicht möglich ist, sonst true.

`removeModel(modelName: QString, pluginName: QString): bool`

- Löscht das bereits existierende Modell mit dem angegebenen Namen des Modells beim angegebenen Plugin. Gibt true zurück, falls das Model erfolgreich gelöscht wurde, und false, falls das Verfahren nicht möglich ist.

`getAugmentationPreview(pluginName : QString, inputPath : QString) : bool`

- Zeigt eine Preview des von dem Plugin mit den aktuellen Einstellungen vorgenommenen Data Augmentation an.

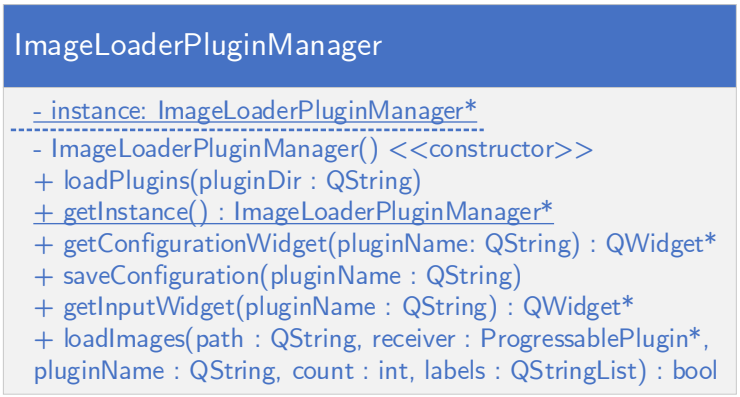
`train(pluginName: QString, modelName: QString, dataSetPath: QString receiver : Progressable*): TrainingResult`

- Startet das Trainingsprozess des Models mit dem angegebenen Name. Gibt anschließend das Ergebnis des Trainings zurück.

`classify(pluginName: QString, inputImagePath: QString, modelName: QString, receiver : Progressable*): ClassificationResult`

- Startet das Klassifizierungsprozess anhand der angegebenem Daten. Gibt das Ergebnis der Klassifizierung zurück.

5.2.4 ImageLoaderPluginManager



Klassenbeschreibung

- Anwendungszentrale Verwaltung der Plugins, die das Herunterladen von Bildern aus dem Internet ermöglichen.
- Die Klasse implementiert die Schnittstelle `PluginManager`.
- Der Einsatz des *Singleton*-Entwurfsmuster stellt sicher, dass nur eine Instanz vom `PictureLoaderPluginManager` existiert.

Attribute

- Statische *Singleton*-Instanz
- Referenz zum `ImageLoaderPlugin`

Konstruktoren

`ImageLoaderPluginManager()`

- Privater Konstruktor des Singleton-Objektes.

Methoden

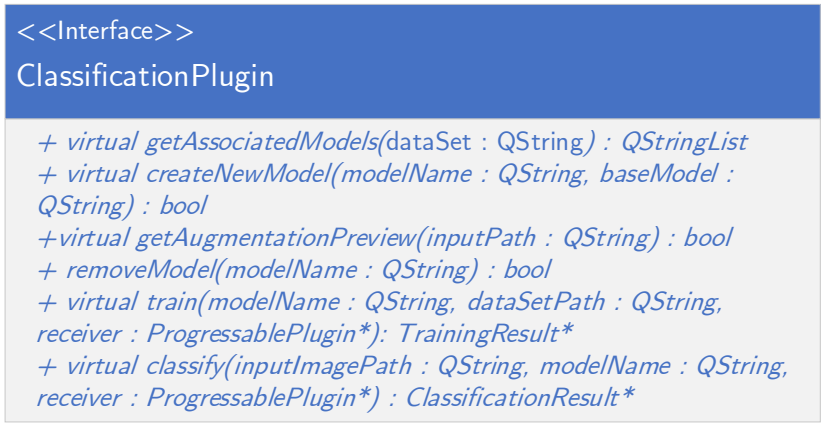
`getInstance() : ImageLoaderPluginManager`

- Gibt eine Referenz zur einzigen Instanz von PictureLoaderPluginManager zurück und legt diese Instanz an, falls noch keine existiert.

`loadImages(path : QString, receiver : ProgressablePlugin*, pluginName : QString, count : int, labels : QStringList) : bool`

- Lädt die Bilder herunter, falls es möglich ist, und gibt true zurück. Sonst gibt false aus.

5.2.5 ClassificationPlugin



Klassenbeschreibung

- Diese Schnittstelle ermöglicht einen einheitlichen Zugriff auf die Klassifizierungsplugins.
- Die Klasse erweitert das Interface `Plugin`.

Methoden

`virtual get associatedModels(dataSet : QString): QStringList`

- Gibt eine Liste mit den Namen aller mit dem Plugin verbundenen Modelle auf dem gegebnen Datensatz zurück.

`virtual createNewModel(modelName: QString, baseModel: QString): bool`

- Legt neues Modell mit den angegebenen Namen des Modells, des Plugins und des Basismodells an. Gibt false zurück, falls das Verfahren nicht möglich ist, sonst true.

`removeModel(modelName: QString): bool`

- Löscht das bereits existierte Modell mit den angegebenen Namen des Models. Gibt true zurück, falls das Model erfolgreich gelöscht wurde, und false, falls das Verfahren nicht möglich ist.

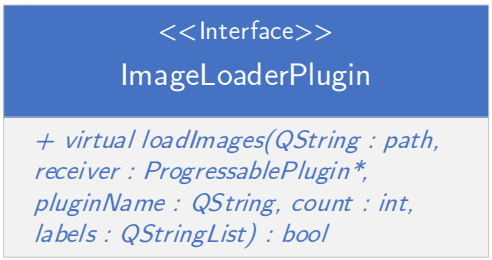
`virtual train(modelName : QString, dataSetPath : QString, receiver : ProgressablePlugin*): TrainingResult*)`

- Startet den Trainingsprozess mithilfe eines Modells und eines Datensatzes. Gibt das Ergebnis des Trainings zurück.

`virtual classify(inputImagePath : QString, modelName :QString, receiver : ProgressablePlugin*) : ClassificationResult*)`

- Startet das Klassifizierungsprozess anhand der angegebenem Daten. Gibt das Ergebnis der Klassifizierung zurück.

5.2.6 ImageLoaderPlugin



Klassenbeschreibung

〉 Diese Schnittstelle ermöglicht einen einheitlichen Zugriff auf die Bildsammlerplugins.

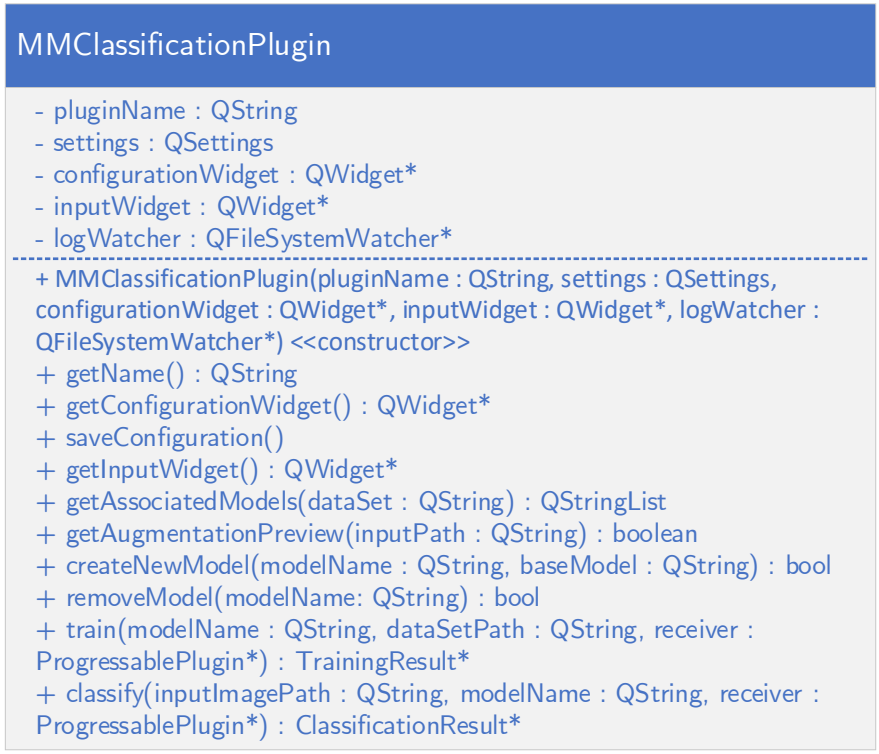
〉 Die Klasse erweitert das Interface `Plugin`.

Methoden

```
virtual loadImages(QString : path, receiver :
ProgressablePlugin*, pluginName : QString, count : int, labels
: QStringList) : bool
```

- Lädt die mit count spezifizierte Anzahl an Bildern mit den angegeben Labels in den mit path spezifizierten Ordner herunter, sofern dies möglich ist. Bei Erfolg wird true zurück, andernfalls wird false zurück gegeben.

5.2.7 MMClassificationPlugin



Klassenbeschreibung

〉 Diese Klasse ist dafür zuständig, neue Modellen anzulegen oder bereits existierende zu löschen, sowie anhand von *MMClassification* einen Trainingsprozess oder einen Klassifizierungsprozess mit dem Modell auszuführen.

〉 Die Klasse implementiert die Schnittstelle `ClassificationPlugin`

Methoden

In dieser Klasse werden lediglich die Methoden implementiert, deren Funktionalität bereits in den Klassen `ClassificationPlugin` und `Plugin` beschrieben wurde.

5.2.8 FlickrPlugin

Klassenbeschreibung

〉 Diese Klasse ist dafür zuständig, das Herunterladen der Bilder aus dem Internet anhand der *Flickr* API auszuführen.

〉 Die Klasse implementiert die Schnittstelle `PictureLoaderPlugin`.

Konstruktoren

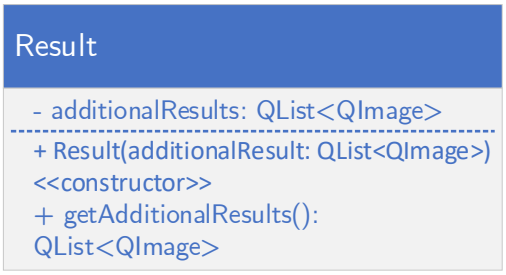
```
FlickrPlugin(pluginName : QString, settings : QSettings,
configurationWidget : QWidget*, inputWidget : QWidget*)
```

- `Instanziiert die FlickrPlugin Klasse.`

Methoden

In dieser Klasse werden lediglich die Methoden implementiert, deren Funktionalität bereits in den Klassen `PictureLoaderPlugin` und `Plugin` beschrieben wurden.

5.2.9 Result



Klassenbeschreibung

〉 Diese Klasse ist dafür zuständig, die Ergebnisse, die durch Plugin spezifische Metriken entstanden sind, an die Rahmenanwendung übermittelt und dort abgespeichert werden können.

Konstruktor

```
+ Result(additionalResult: QList<QImage>)
```

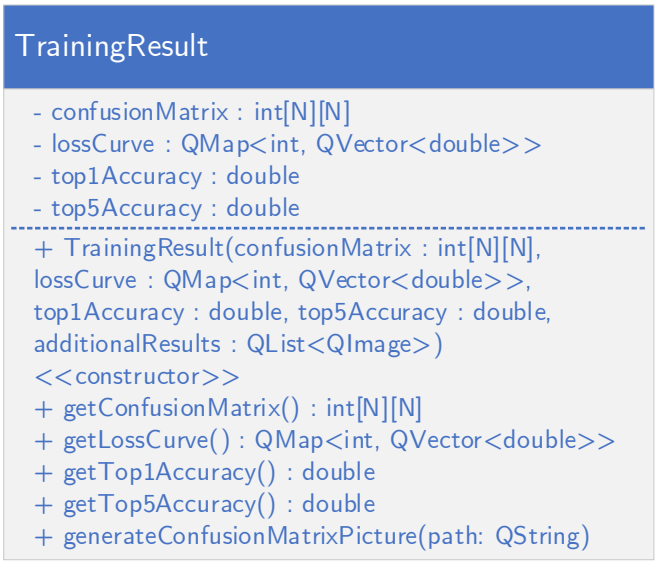
- Instanziiert die Result Klasse.

Methoden

```
+ getAdditionalResults(): QList<QImage>
```

- Gibt Plugin spezifische Ergebnisse in Form einer Liste von QImage Objekten zurück.

5.2.10 TrainingResult



Klassenbeschreibung

〉 Diese Klasse ist dafür zuständig, die einzelnen Ergebnisse des Trainingsprozesses zur Weitergabe an die Rahmenanwendung zu halten und dadurch eine einfachere Visualisierung und Speicherung dieser Ergebnisse über die Rahmenanwendung zu ermöglichen.

〉 Erbt von der Klasse `Result`.

Konstruktor

```
TrainingResult(aconfusionMatrix : int[N][N],
lossCurve : QMap<int, QVector<double>>,
top1Accuracy : double, top5Accuracy :
double, additionalResults : QList<QImage>)
```

- Instanziiert die TrainingResult Klasse.

Methoden

+ getConfusionMatrix(): int[N][N]

- Erstellt eine Konfusionsmatrix anhand des Ergebnisses des ausgeführten Trainingsprozesses und gibt diese zurück.

+ getLossCurve(): QMap<int, QVector<double>>

- Erstellt eine Verlustkurve anhand des Ergebnisses des ausgeführten Trainingsprozesses und gibt diese zurück.

+ getTop1Accuracy(): double

- Gibt die Top 1 Genauigkeit des Modells zurück.

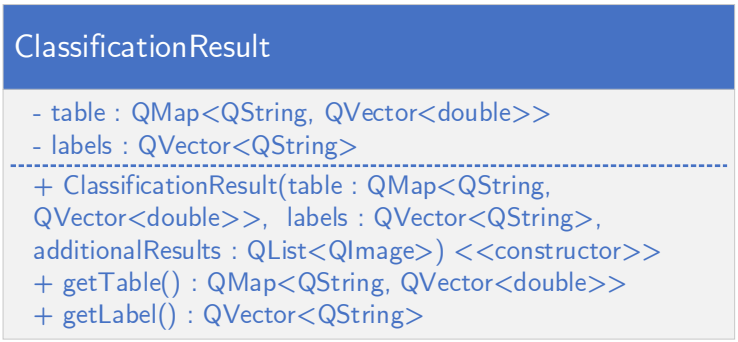
+ getTop5Accuracy(): double

- Gibt die Top 5 Genauigkeit des Modells zurück.

+ generateConfusionMatrixPicture(path: QString)

- Generiert mithilfe zusätzlicher Funktionalität aus Python eine Kunfusionsmatrix und speichert sie im Verzeichnis path.

5.2.11 ClassificationResult



Klassenbeschreibung

〉 Diese Klasse ist dafür zuständig, die einzelnen Ergebnisse einer Klassifizierung zur Weitergabe an die Rahmenanwendung zu halten und dadurch eine einfachere Visualisierung und Speicherung dieser Ergebnisse über die Rahmenanwendung zu ermöglichen.

〉 Erbt von `Result`.

Konstruktor

```
ClassificationResult(table : QMap<QString, QVector<double>>,  labels : QVector<QString>, additionalResults : QList<QImage>)
```

- Instanziiert die ClassificationResult Klasse.

Methoden

```
+ getTable(): QMap<QString,QVector<double>>
```

- Gibt eine Tabelle mit den Ergebnissen der Klassifizierung für die einzelnen Eingabebilder zurück.

```
+ getLabel(): QVector<QString>
```

- Gibt eine Liste mit den Labels zurück mit den die Eingabebilder klassifziert wurden.

6 Serveranwendung

Unser Projekt ist – gemäß den Wunschkriterien aus dem Pflichtenheft – auf einem Linux Server ferngesteuert ausführbar. Zur Illustration dienen die folgenden Netzwerkdiagramme.

6.1 Applikation auf lokalem Computer

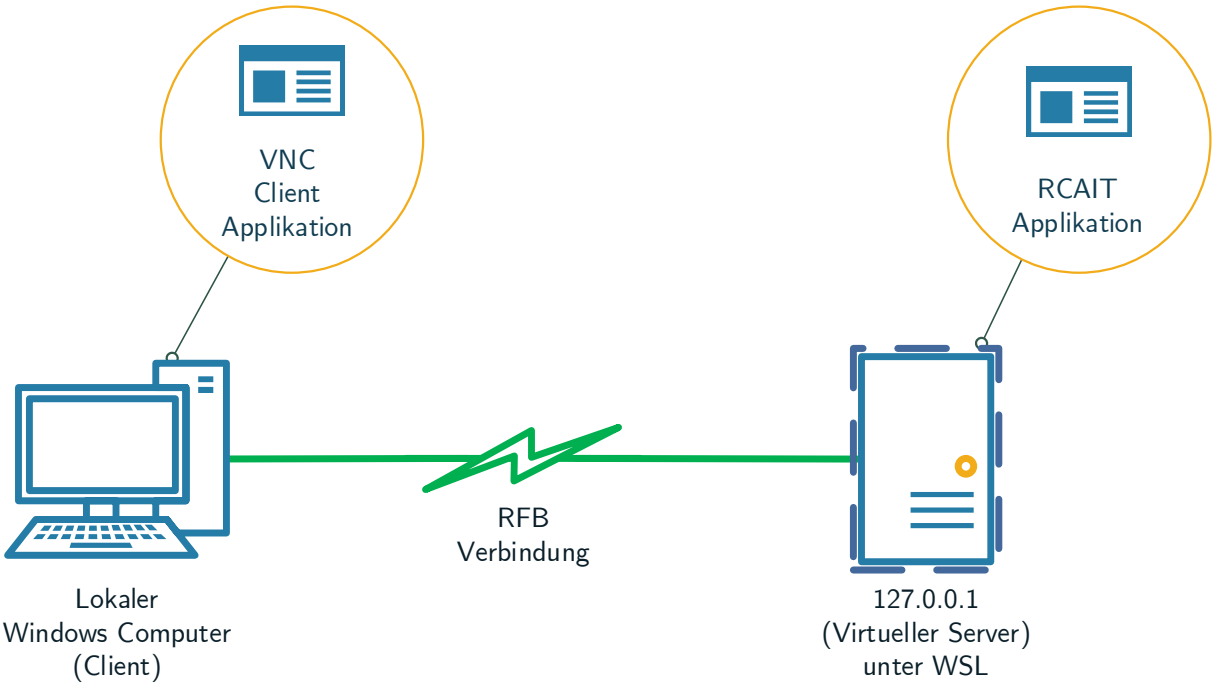


Abbildung 7 : Verbindungsaufbau innerhalb eines einzelnen Computers

Dieses Szenario wird oft für Testzwecke verwendet – beispielsweise, wenn kein realer Server zur Verfügung steht. Die Anwendung wird dabei in Qt für Linux kompiliert, um Unterstützung für das VNC Server Modul zu erhalten. Dieses ist unter Windows nicht enthalten.
Deshalb wird das *Windows Subsystem for Linux* (kurz WSL) verwendet.

Da sich der virtuelle Server auf dem gleichen Computer befindet, sind die Reaktionszeiten der Verbindung kurz und die Geschwindigkeit besonders hoch.

6.2 Applikation auf Remote Server

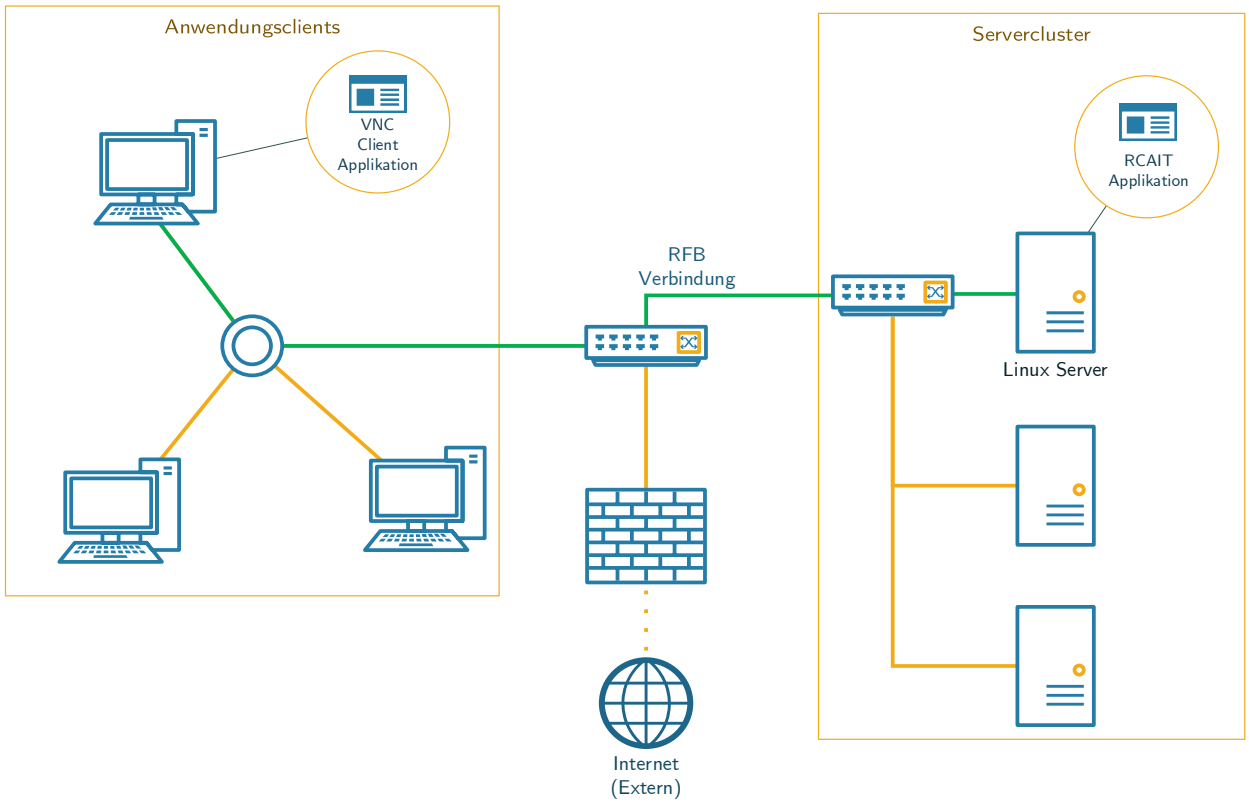


Abbildung 8: Verbindungsaufbau über Intranet

In diesem Szenario wird eine beispielhafte Struktur des Fraunhofer IOSB Intranets dargestellt. Es ist auch kein Problem, wenn Server und Client nicht räumlich beieinander liegen, solange eine Verbindung im Intranet besteht. Andernfalls kann z.B. ein VPN verwendet werden, um das zu gewährleisten.

VNC arbeitet nach dem Client-Server-Modell. Nach der Kompilierung wird vom Server nun auf eine eingehende Verbindung eines Clients unter einem vorgegebenen Port gewartet. Mit einem entsprechenden VNC-Client lässt sich nun die Anwendung in einem Fenster mit Bildschirmausgabe sowie Eingabemöglichkeiten auf dem eigenen PC nutzen.

VNC hat zudem noch weitreichende Vorteile, so kann die Verbindung ohne Probleme mit einem Passwort TLS-verschlüsselt werden. Dies ist ein großer Vorteil für Netzwerke mit vielen fremden Geräten. Zudem ist das verwendete *Remote Framebuffer Protokoll* (kurz RFB) quelloffen verfügbar und im Gegensatz zu anderer Fernwartungssoftware, plattformunabhängig benutzbar. Es kann daher auf praktisch jedem Computer, oder sogar Smartphone, ausgeführt werden, ohne das Programm anzupassen. Somit wird hier eine große Flexibilität geboten.

7 Ablaufbeschreibungen

Um das Zusammenspiel der einzelnen Klassen in unserem Projekt besser verstehen zu können, gibt es nachfolgend ein paar UML Sequenzdiagramme.

7.1 Ausführen eines Trainings

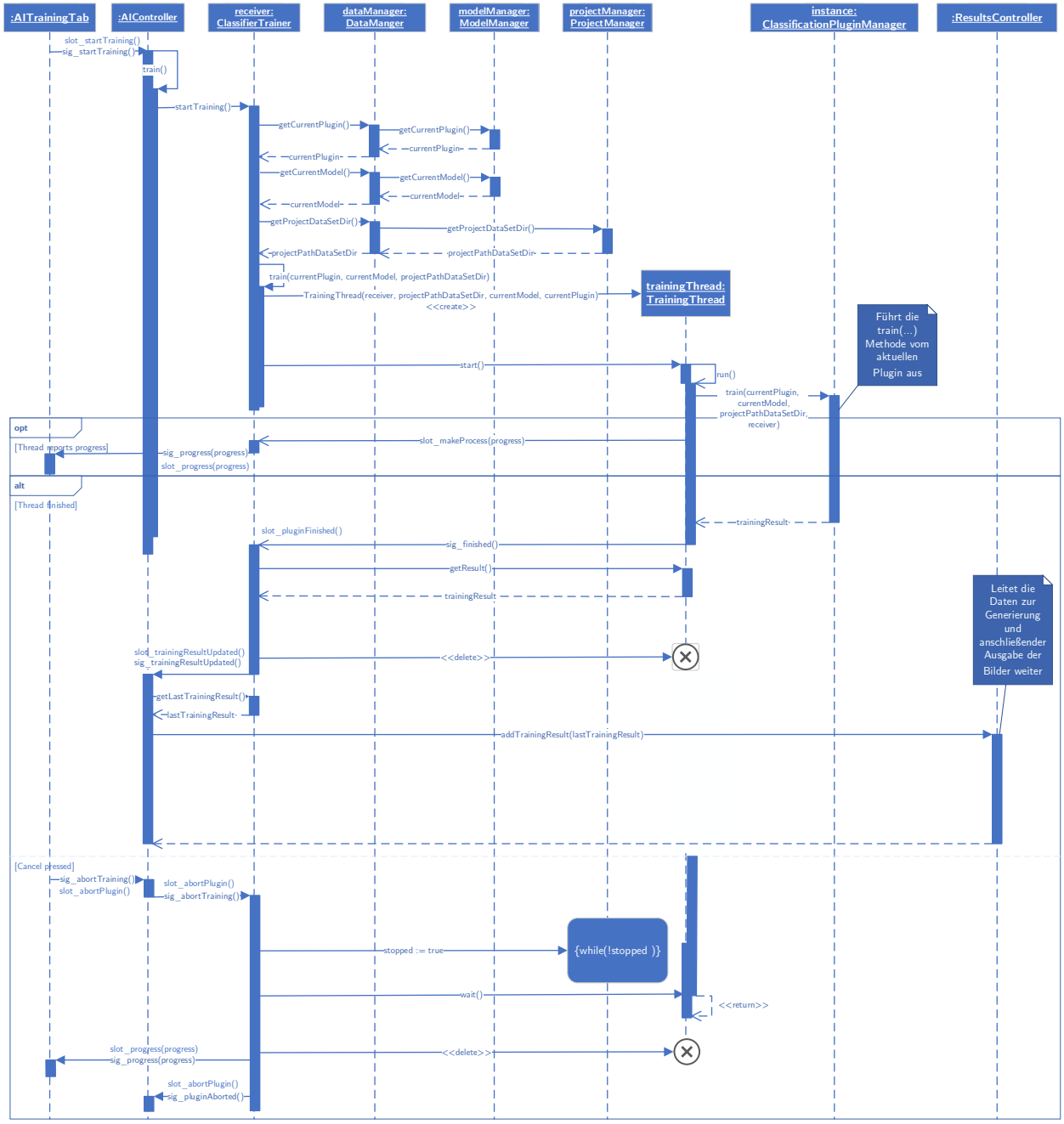


Abbildung 9: Sequenzdiagramm zu `sig_startTraining()`

7.2 Vergleichen von Modellen

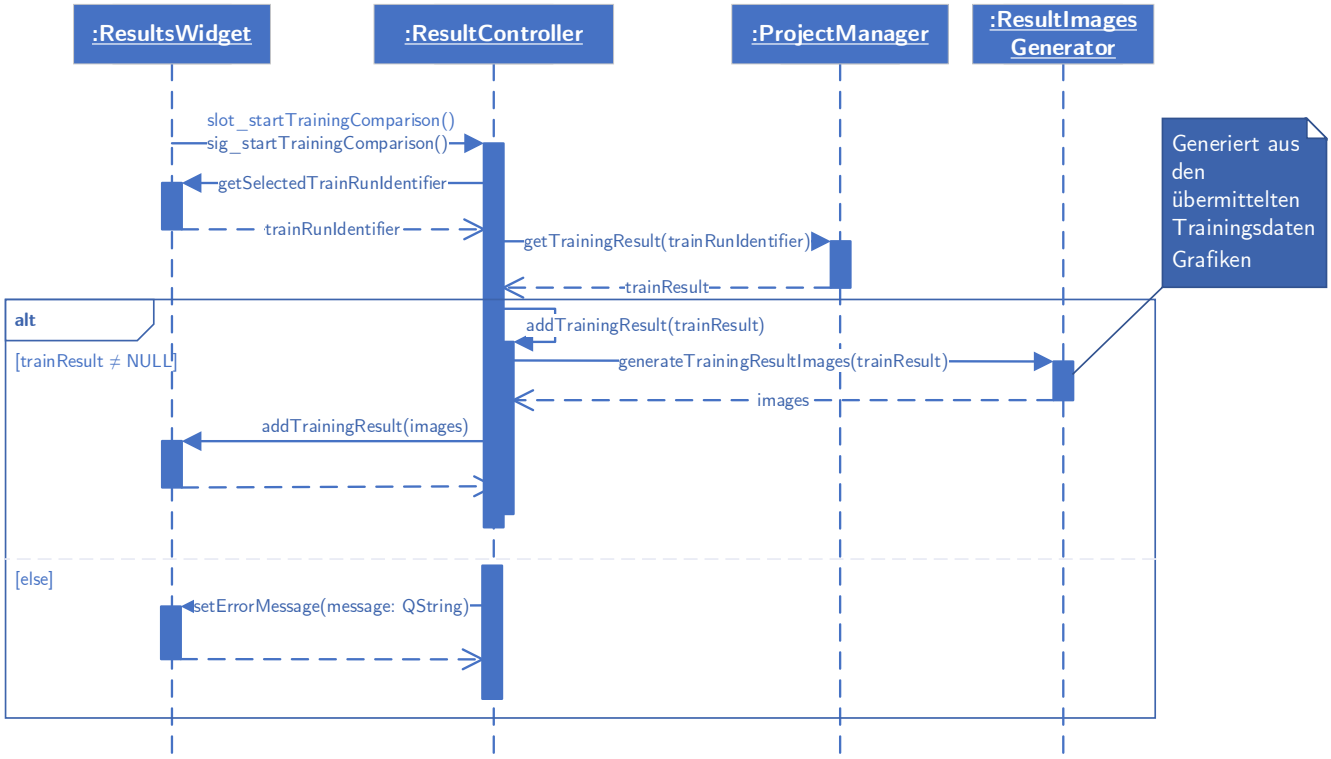


Abbildung 10: Sequenzdiagramm zu `sig_startTrainingComparison()`

7.3 Herunterladen von Bildern

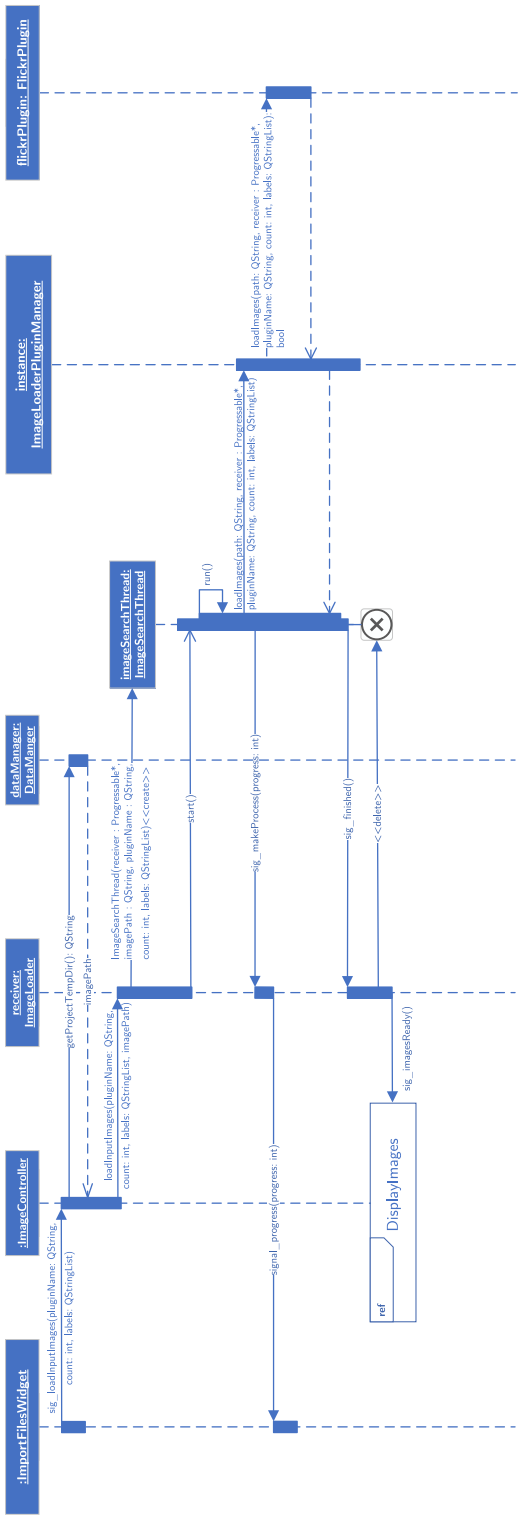


Abbildung 11: Sequenzdiagramm zu `sig_loadInputImages()`

8 Änderungen zum Pflichtenheft

In Bezug auf das Pflichtenheft gibt es folgenden Änderungen:

- Das Programm wird sowohl deutsch- als auch englischsprachig ausgeliefert
- Weitere Sprachen können hinzugefügt werden, ohne den bestehenden Code zu verändern
- Optional können auch mehr als zwei Klassifikationsmodelle verglichen werden
- Der Results-Tab wurde dazu wie folgt aktualisiert:

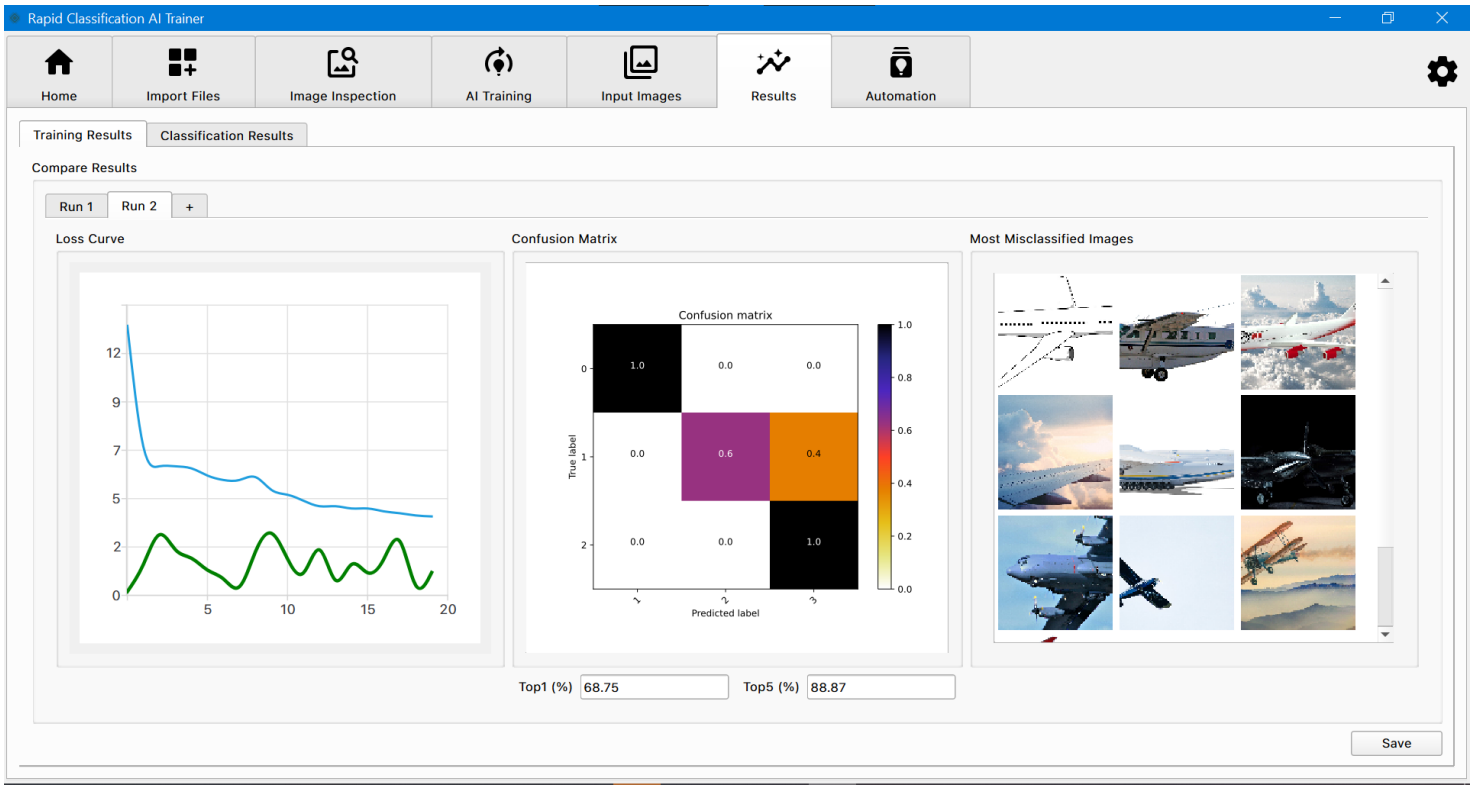


Abbildung 9 : Neuer Results Tab

9 Glossar

Basismodell bezeichnet ein vortrainiertes Modell, das dem Benutzer zur Erstellung eines spezifischeren Modells zur Verfügung steht. Es wird von einem Plugin in die Anwendung eingespeist und kann bei dem Hinzufügen eines Modells als Grundlage ausgewählt werden. Durch diesen Schritt des Hinzufügens entsteht dann ein Duplikat, dass mit dem Trainingsvorgang verändert werden kann. Das Basismodell ändert sich dementsprechend nicht. Beispielsweise liefert unsere Anwendung ein auf Image Net vortrainiertes Modell als Basismodell aus.

C++ Eine von der ISO genormte Programmiersprache; ermöglicht sowohl die effiziente und maschinennahe Programmierung als auch eine Programmierung auf hohem Abstraktionsniveau.

Deep Learning (tiefes Lernen) bezeichnet eine Methode des maschinellen Lernens, die künstliche neuronale Netze (KNN) mit zahlreichen Zwischenschichten zwischen Eingabeschicht und Ausgabeschicht einsetzt und dadurch eine umfangreiche innere Struktur herausbildet.

Git eine freie Software zur verteilten Versionsverwaltung von Dateien, die durch Linus Torvalds initiiert wurde.

GUI Bezeichnung für Graphische Nutzeroberfläche.

Flickr kommerzieller Onlinedienst mit Community-Elementen, der es Benutzern erlaubt, digitale und digitalisierte Bilder sowie kurze Videos von maximal drei Minuten Dauer mit Kommentaren und Notizen auf die Website zu laden und so anderen Nutzern zugänglich zu machen.

Klassifikator Algorithmus, speziell in der Mustererkennung eine mathematische Funktion, die einen Merkmalsraum auf eine Menge von Klassen abbildet.

Konfusionsmatrix Tabellenlayout, das die Visualisierung der Leistung eines Algorithmus ermöglicht; überprüft, ob die Prognose einer Klassifikation richtiger-/fälschlicherweise wahr oder falsch ist.

Künstliche Intelligenz (KI) auch artifizielle Intelligenz (AI) ein Versuch, bestimmte Entscheidungsstrukturen des Menschen nachzubilden, indem z. B. ein Computer so gebaut und programmiert wird, dass er relativ eigenständig Probleme bearbeiten kann.

Labeling Annotieren der Bilddaten mit Suchworten als Beschriftung (engl. *label*) für nachgelagerte Verarbeitungsschritte.

Nutzer Person die das Rapid Classification AI Programm nutzt.

Objekterkennung Verfahren zum Identifizieren bekannter Objekte innerhalb eines Objektraums mittels optischer, akustischer oder anderer physikalischer Erkennungsverfahren.

Plugin optionale Software-Komponente, die eine bestehende Software erweitert bzw. verändert.

Projekt Ein Bilddatensatz in Verbindung mit den darauf trainierten Modellen.

Qt Anwendungsframework und GUI-Toolkit zur plattformübergreifenden Entwicklung von Programmen und grafischen Benutzeroberflächen.

Sammlerplugin Eine konkrete Art von Plugin das die Möglichkeit bietet Bildermengen anhand von Suchbegriffen automatisch herunterzuladen.

Server ein Rechner, der für andere in einem Netzwerk mit ihm verbundene Systeme bestimmte Aufgaben übernimmt und von dem diese ganz oder teilweise abhängig sind.

Training des Deep-Learning-Modells «Fütterung» des Modells mit großen Trainingsdaten, um den Algorithmus zur Lösung eines Problems zu trainieren.

Validierung des Deep-Learning Modells Auswertung eines bereits annotierten Bilder Satzes durch ein Deep-Learning-Modell, mit dem Ziel die Exaktheit des Modells quantifizieren zu können.

VNC Virtual Network Computing. Dieses wird für das Fernsteuern eines Programms auf einem entfernten Computer verwendet.

Signale & Slots Durch Qt bereitgestellte Möglichkeit, Sender und Empfänger zwischen Klassen zu verwenden. Signale werden dabei vom Sender ausgestrahlt und von Empfängerobjekten durch einen zugehörigen Slot empfangen.

Entwurfsmuster Bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme.

Fassade Strukturelles Entwurfsmuster zur Vereinheitlichung komplexer Klassen hinter einer vereinfachten Schnittstelle.

Singleton Erzeugendes Entwurfsmuster zur Sicherstellung, dass von einer Klasse genau eine Instanz existiert.

Beobachter Entwurfsmuster, das eine 1 zu N Abhängigkeit erzeugt, sodass bei einer Zustandsänderung alle abgängigen Objekte benachrichtigt werden.