Praxis der Softwarentwicklung

Designbook

Neural Network based Image Classification System on Heterogeneous Platforms

from

Team 2
Dimitrov, Drehwald, Guneshka, Häring, Stangel

Contents

1 Introduction

This document describes the design of the PSE project "Neural Network based Image Classification System on Heterogeneous Platforms" by Team 2. It will explain the different modules and how they work together. This document will be the base for the next phase (the implementation).
The design is made with UML class diagrams and the explanation is a code documentation with which the implementation can be done blindly. For further clearance we used UML activity and sequence diagrams.
The responsible person for this phase is Johannes Häring.

2 PackageDesign

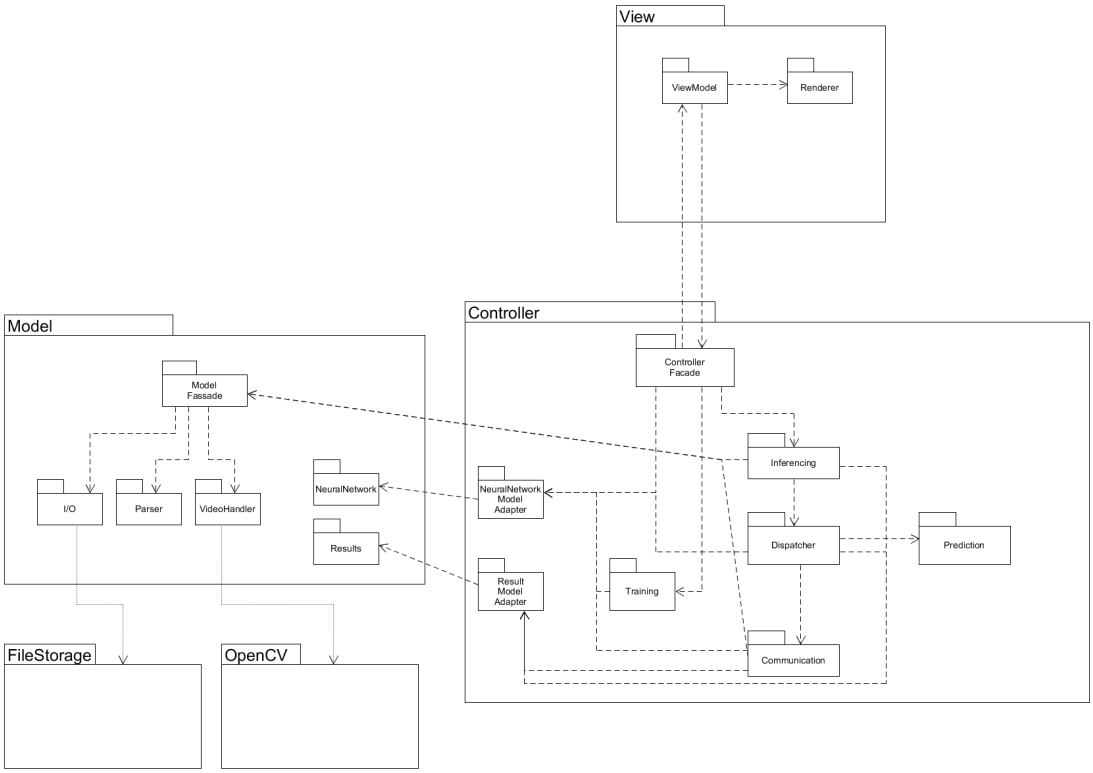


Figure 1: Package design of our software

Our software is designed with the Model-View-Controller pattern.
Through this design we are able to swap one of the parts with another one which allows to flexible adjust our software to the current needs.
Between the top packages Model, View and Controller are four accumulated connections.
The packages interact mainly through facades.
The controller module holds all the program logic. It controls the Model module and View module and calls needed methods.

3 Package Description: ViewModule

This is the GUI package to interact with. It is only for representation and holds no logic components. When buttons are clicked or other input happens the onClick()-method of those just calls the ControlFacade which calls the corresponding EventHandler.
The whole GUI is done with the QT library.

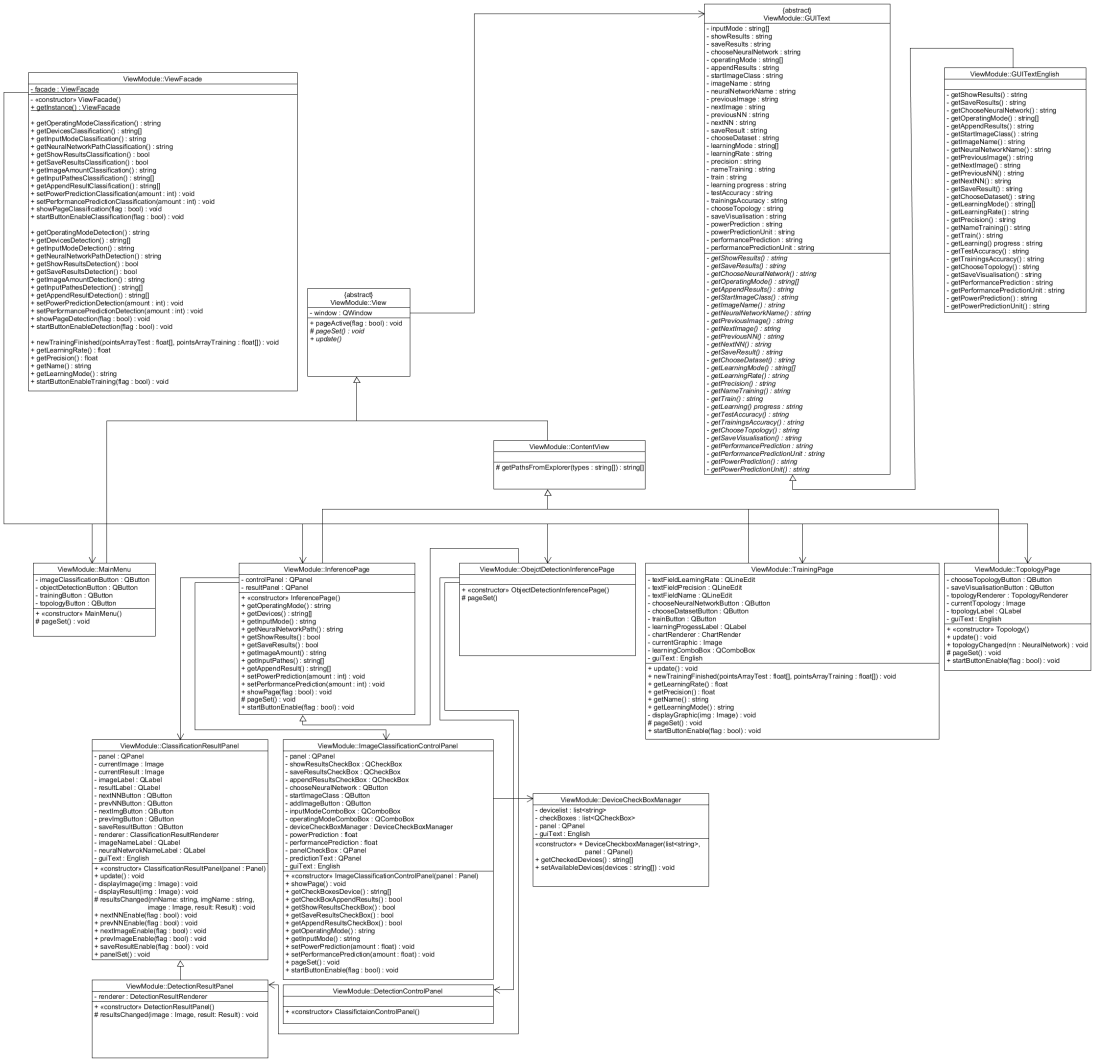


Figure 2: Package design of the view module

3.1 Class Description: View

This is an abstract class, that defines the main parts of a page.
Attributes:
- window : QWindow
The window to display.

- text : GUIText
The page gets all the messages and texts from this class to be easily changeable.

Methods:

+ pageActive(flag : bool) : void
Sets the window active according to the flag.

 # pageSet() : void
This is an abstract method which sets the whole page view up.

+ update()
This is an abstract method that is used to update the View if a new event happens.

3.2 Class Description: MainMenu

MainMenu extends View
This class is used to to show and control the main page of the software.

Attributes:

- imageClassificationButton : QButton
- objectDetectionButton : QButton
- trainingButton : QButton
- topologyButton : QButton

Methods:

+ «constructor» MainMenu()
This is a constructor that sets the whole page by first setting the language and then using the pageSet() method.

pageSet() : void
This method sets up the page of the main menu.

3.3 Class Description: ContentView

ContentView extends View
This class is used to simplify the access for all of the Views to the user's data from his File Explorer.

Methods:

getPathsFromExplorer(types: string[]) : string[]
This method is used to get data from the user's personal computer by opening a file explorer. The parameter types carries the information what kind of extensions are allowed to be seen in the file explorer.

3.4 Class Description: InferencePage

InferencePage extends ContentView
This class represents the ImageClassification GUI. It contains two panels. One to handle the input choices and one to display the results.

Attributes:

- controlSection : ImageClassificationControlPanel
- resultSection : ClassificationResultPanel
- controlPanel : QPanel
The panel the controlSection should work in.

- resultPanel : QPanel
The panel the resultSection should work in.

Methods:

+ «constructor» InferencePage()
This is a constructor that sets the whole page ImageClassificationPage.

+ getOperatingMode() : string
This method returns ImageClassificationControlPanel.getOperatingMode()

+ getDevices() : string[]
This method returns ImageClassificationControlPanel.getDevices()

+ getInputMode() : string
This method returns ImageClassificationControlPanel.getInputMode()

+ getNeuralNetworkPath() : string
This method returns ImageClassificationControlPanel.getNeuralNetworkPath()

+ getShowResults() : bool
This method returns ImageClassificationControlPanel.getShowResults()

+ getSaveResults() : bool
This method returns ImageClassificationControlPanel.getSaveResults()

+ getImageAmount() : string
This method returns ImageClassificationControlPanel.getImageAmount()

+ getInputPathes() : string[]
This method returns ImageClassificationControlPanel.getInputPathes()

+ getAppendResult() : string[]
This method returns ImageClassificationControlPanel.getAppendResult()

+ showPage() : void
This method is used to show the Results and ImageClassification pages of the software using their showPage method.

3.5 Class Description: ImageClassificationControlPanel

This class is used to show the Image Classification Control GUI of the software.

Attributes:

- panel : QPanel - showResultsCheckBox : QCheckBox
- saveResultsCheckBox : QCheckBox
- appendResultsCheckBox : QCheckBox
- chooseNeuralNetwork : QButton
- startImageClass : QButton
- addImageButton : QButton
- inputModeComboBox : QComboBox
- operatingModeComboBox : QComboBox
- deviceCheckBoxManager : DeviceCheckBoxManager
The deviceCheBoxManager handles the checkboxes for the devices.

- panelCheckBox : QPanel
The panel where the checkboxes for the devices lay in.

- predictionText : QPanel
The predicted values are printed in here.

- guiText : GUITextEnglish

Methods:

+ «constructor» ImageClassificationControlPanel(panel : QPanel)
This is a constructor that sets the whole panel using the panelSet() - method.

+ getCheckBoxesDevice() : string[]
This method returns deviceCheckBoxManager.getCheckedDevices()

+ getCheckBoxAppendResults() : bool
This method returns value of the appendResultsCheckBox.

+ getShowResultsCheckBox() : bool
This method returns value of the showResultsCheckBox.

+ getSaveResultsCheckBox() : bool
This method returns value of the saveResultsCheckBox.

+ getAppendResultsCheckBox() : bool
This method returns value of the appendResultsCheckBox.

+ getOperatingMode() : string
This method returns value of the operatingModeComboBox.

+ getInputMode() : string
This method returns value of the inputModeComboBox.

+ showPage() : void
This method is used to show the Main Menu page of the software.

- panelSet() : void
This method sets up the page.

3.6 Class Description: DeviceCheckBoxManager

This class is responsible to show all the Device checkboxes and change them dynamically.

Attributes:

- devicelist : list<string>
- checkBoxes : list<QCheckBox>
- panel : QPanel
- guiText : English

Methods:

+ «constructor» DeviceCheckboxManager(list<string>, panel : QPanel)

+ getCheckedDevices() : string[]
Returns the value of checkBoxes.

+ setAvailableDevices(devices : string[]) : void
sets which Devices are available for use.

3.7 Class Description: ClassificationResultPanel

This class is used by the InferencePage to control the view of the Image Classification Result Panel of the software.

Attributes:

- panel : QPanel
The panel to display everything. - currentImage : Image
- currentResult : Image
- imageLabel : QLabel
- resultLabel : QLabel
- nextNNButton : QButton
- prevNNButton : QButton
- nextImgButton : QButton
- prevImgButton : QButton
- saveResultButton : QButton
- renderer : ClassificationResultRenderer
- imageNameLabel : QLabel

- neuralNetworkNameLabel : QLabel
- guiText : GUITextEnglish

Methods:
+ «constructor» ClassificationResultPanel(panel : QPanel)
This is a constructor that sets the whole panel using the panelSet() method.

+ update() : void
This is method is used to update the view. Change the currentImage and currentResults shown using the displayImage(img : Image) and displayResult(img : Image) methods.

- displayImage(img : Image) : void
This method displays an image in the panel. It is used to display the main image.

- displayResult(img : Image) : void
This method displays an image in the panel. It is used to display the results.

resultsChanged(nnName: string, imgName : string, image : Image, result: Result) : void
This method is used to notify when the results are done and ready to be pulled. The method then calls the renderer to render the results but it does not update the GUI yet.

+ nextNNEnable(flag : bool) : void
This method enables/disables the nextNNButton.

+ prevNNEnable(flag : bool) : void
This method enables/disables the prevNNButton.

+ nextImageEnable(flag : bool) : void
This method enables/disables the nextImageButton.

+ prevImageEnable(flag : bool) : void
This method enables/disables the prevImageButton.

+ saveResultEnable(flag : bool) : void
This method enables/disables the saveResultButton.

+ panelSet() : void
This method sets up the panel.

3.8 Class Description: TopologyPage

Topology extends ContentView
This class is used to show and control the Topology page of the software.

Attributes:

- chooseTopologyButton : QButton
- saveVisualisationButton : QButton
- topologyRenderer : TopologyRenderer
- currentTopology : Image
- topologyLabel : QLabel
The label that holds the topology image. - guiText : GUITextEnglish

Methods:
+ «constructor» TopologyPage()
This is a constructor that sets the whole page using the pageSet() method.

+ update() : void
This method changes the currentTopology showed in the view.

+ topologyChanged(nn : NeuralNetworkAdapter) : void
This method is used by the Controller to notify when the results are done and ready to be pulled. The method then calls the renderer to render the results.

pageSet() : void
This method sets up the page.

3.9 Class Description: TrainingPage

Training extends ContentView
This class is used to show and control the Training page of the software.

Attributes:

- textFieldLearningRate : QLineEdit
- textFieldPrecision : QLineEdit
- textFieldName : QLineEdit
- chooseNeuralNetworkButton : QButton
- chooseDatasetButton : QButton

- trainButton : QButton
- learningProgessLabel : QLabel
- chartRenderer : ChartRender
- currentGraphic : Image
- learningComboBox : QComboBox
- guiText : GUIText

Methods:
+«constructor» TrainingPage() This is a constructor that sets the whole page using the pageSet() method.

+ update() : void
This method is used to know when the currentGraphic has to be changed in the view.

+ newTrainingFinished(pointsArrayTest : float[], pointsArrayTraining : float[]) : void
This method is used by the Controller to notify when the results are done and ready to be pulled. The method then calls the renderer to render the results.

+ getLearningRate() : float
This method returns value of textFieldLearningRate.

+ getPrecision() : float
This method returns value of textFIeldPrecision.

+ getName() : string
This method returns value of textFieldName.

+ getLearningMode() : string
This method returns value of learningComboBox

- displayGraphic(img : Image) : void
This method changes the displayed graphic in the view.

+ pageSet() : void
This method sets the page up.

3.10 Class Description: DetectionControlPanel

Extends ImageClassificationControPanel
This class is used to control the object detection panel of the software. The onClick()-functions of the buttons are set differently to call the right method in the control facade.

Methods:

+ «constructor» DetectionControlPanel(panel : QPanel)

- panelSet() : void
Sets up the panel.

3.11 Class Description: DetectionResultPanel

Extends ClassificationResultPanel
This class is used to show and control the Object Detection panel of the software. The onClick()-functions of the buttons are set differently to call the right method in the control facade.

Attributes:

- renderer : DetectionResultRenderer

Methods:

+ «constructor» DetectionResultPanel(panel : QPanel)
This is a constructor that sets the whole page using the panelSet() method.

resultsChanged(result: Result) : void
This method is used by the Controller to notify when the results are done and ready to be rendered. The method then calls the renderer to render the results.

3.12 Class Description: ObjectDetectionInferencePage

ObjectDetectionInferencePage extends InferencePage
This class displays the detection page similar to the detection page. Just different control panels and result panels are created.

Methods:

+ «constructor» ObjectDetectionInferencePage()
The constructor creates two panels and then the resultsection and control section.

pageSet() : void

3.13 Class Description: GUIText

This class should define an interface that gives all text in the GUI.
If needed, the language of the labels can be changed easily.

Attributes:
- inputMode : string[]
- showResults : string
- saveResults : string
- chooseNeuralNetwork : string
- operatingMode : string[]
- appendResults : string
- startImageClass : string
- imageName : string
- neuralNetworkName : string
- previousImage : string
- nextImage : string
- previousNN : string
- nextNN : string
- saveResult : string
- chooseDataset : string
- learningMode : string[]
- learningRate : string
- precision : string
- nameTraining : string
- train : string

- learning progress : string
- testAccuracy : string
- trainingsAccuracy : string
- chooseTopology : string
- saveVisualisation : string

Methods:

The following methods are abstract and used for later implementation of the concrete language.

- getShowResults() : string
- getSaveResults() : string
- getChooseNeuralNetwork() : string
- getOperatingMode() : string[]
- getAppendResults() : string
- getStartImageClass() : string
- getImageName() : string
- getNeuralNetworkName() : string
- getPreviousImage() : string
- getNextImage() : string
- getPreviousNN() : string
- getNextNN() : string
- getSaveResult() : string
- getChooseDataset() : string
- getLearningMode() : string[]
- getLearningRate() : string
- getPrecision() : string
- getNameTraining() : string
- getTrain() : string
- getLearning() progress : string
- getTestAccuracy() : string
- getTrainingsAccuracy() : string
- getChooseTopology() : string
- getSaveVisualisation() : string

3.14 Class Description: GUITextEnglish

GUITextEnglish extends GUIText
This class is a concrete implementation of the GUIText Class and implements all the methods with english text.

3.15 Class Description: ViewFacade

This class is used to manage the whole communication between View and the other modules.

Attributes:

- facade : ViewFacade
All the pages:
- mainMenu : MainMenu
- inferencePage : InferencePage
- topology : TopologyPage
- training : TrainingPage
- detectionControlPanel : DetectionControlPanel

Methods:

- «constructor» ViewFacade()
Constructor for the ViewFacade. Creates every page.

+ getInstance() : ViewFacade
This method returns the first instance of the ViewFacade, using the Singleton pattern.

All these methods call the methods of the classification page.
+ getOperatingModeClassification() : string
+ getDevicesClassification() : string[]
+ getInputModeClassification() : string
+ getNeuralNetworkPathClassification() : string
+ getShowResultsClassification() : bool
+ getSaveResultsClassification() : bool
+ getImageAmountClassification() : string
+ getInputPathesClassification() : string[]
+ getAppendResultClassification() : string[]

+ setPowerPredictionClassification(amount : int) : void
+ setPerformancePredictionClassification(amount : int) : void
+ showPageClassification(flag : bool) : void
+ startButtonEnableClassification(flag : bool) : void
+ nextNeuralNetworkButtonEnableClass(flag : bool) : void
+ prevNeuralNetworkButtonEnableClass(flag : bool) : void
+ nextImageButtonEnableClass(flag : bool) : void
+ prevImageButtonEnableClass(flag : bool) : void

All these methods call the methods of the detection page.
+ getOperatingModeDetection() : string
+ getDevicesDetection() : string[]
+ getInputModeDetection() : string
+ getNeuralNetworkPathDetection() : string
+ getShowResultsDetection() : bool
+ getSaveResultsDetection() : bool
+ getImageAmountDetection() : string
+ getInputPathesDetection() : string[]
+ getAppendResultDetection() : string[]
+ setPowerPredictionDetection(amount : int) : void
+ setPerformancePredictionDetection(amount : int) : void
+ showPageDetection(flag : bool) : void
+ startButtonEnableDetection(flag : bool) : void
+ nextNeuralNetworkButtonEnableDetection(flag : bool) : void
+ prevNeuralNetworkButtonEnableDetection(flag : bool) : void
+ nextImageButtonEnableDetection(flag : bool) : void
+ prevImageButtonEnableDetection(flag : bool) : void

All these methods call the methods of the training page.
+ newTrainingFinished(pointsArrayTest : float[], pointsArrayTraining : float[]) : void
+ getLearningRate() : float
+ getPrecision() : float
+ getName() : string
+ getLearningMode() : string
+ startButtonEnableTraining(flag : bool) : void

All these methods call the methods of the topology page.
+ topologyChanged(nn : NeuralNetwork) : void
+ startButtonEnableTopology(flag : bool) : void

+ update() : void
Update() calls the update() - method of every page.

4 Package Description: RendererModule

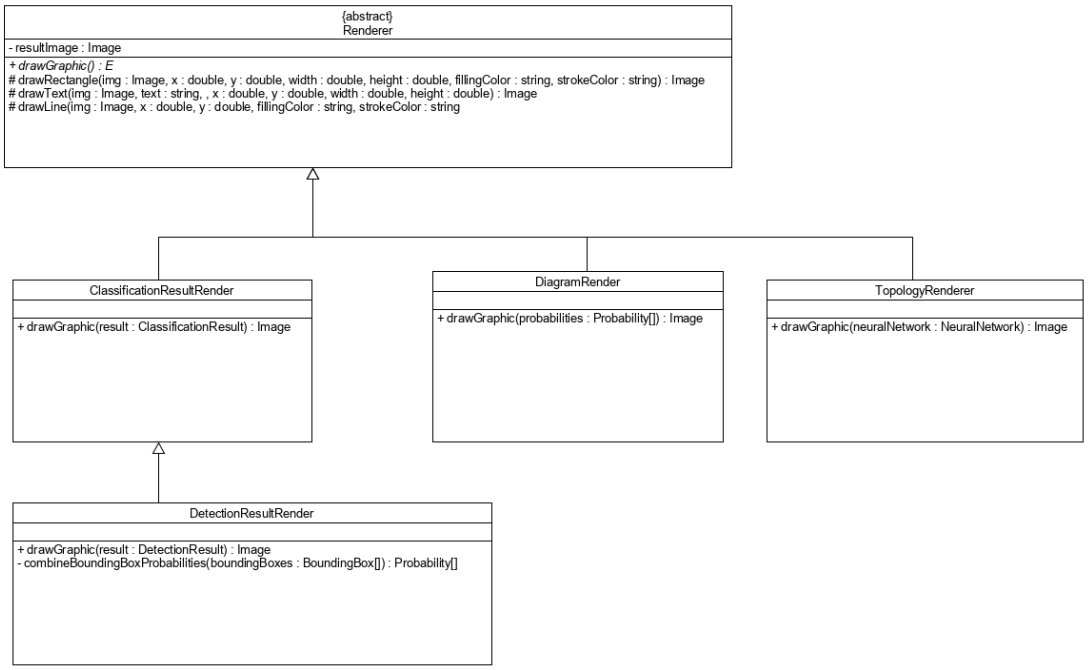


Figure 3: UML diagram of the renderer

The renderer module contains all the renderers for the view.

4.1 Class Description: Renderer

This is an abstract class that defines the basic methods that are needed for rendering and also the structure for the specific renders.

Methods:

+ drawGraphic() : E
This is an abstract method that defines the structure for the inheriting classes.

drawRectangle(img : Image, x : double, y : double, width : double, height : double, fillingColor : string, strokeColor : string) : Image
This method draws a rectangle on a given Image(img parameter). The x and y coordinates show the top left corner of the box. The width and height give the width and height of the box. The fillingColor parameter is used to show in which color to fill the

bounding box and the strokeColor parameter is used to know in which color to draw the corners of the bounding box.

drawText(img : Image, text : string, , x : double, y : double, width : double, height : double) : Image
This method creates a text label on a specific image(img) with given text, position(x,y) and size(width, height).

drawLine(img : Image, x : double, y : double, fillingColor : string, strokeColor : string)
This method draws a line on a specific image(img) by connecting two points

4.2 Class Description: ClassificationResultRenderer

ClassificationResultRenderer extends Renderer
This class is used to render the results for the Image Classification part of the software.

Methods:

+ «constructor» ClassificationResultRenderer()
This is the default constructor.

+ drawGraphic(result : ClassificationResult) : Image
This method is used to draw the graphic with the given results of an ImageClassification.

4.3 Class Description: DiagramRenderer

DiagramRenderer extends Renderer
This class is used to render Diagrams.

Methods:

+ «constructor» DiagramRenderer()
This is the default constructor.

+ drawGraphic(probabilities : Probability[]) : Image
This method draws a graphic by given probabilities(the probabilities parameter) using the drawRectangles(), drawText() and drawLines() methods.

4.4 Class Description: ChartRenderer

ChartRenderer extends Renderer
This class is used to render graphics for the training page of the software.

Methods:
+ «constructor» ChartRenderer()
This is the default constructor.

+ drawGraphic(pointsArrayTest : float[], pointsArrayTraining : float[]) : Image
This method draws a graphic with a given array of points.

4.5 Class Description: TopologyRenderer

TopologyRenderer extends Renderer
This class is used to render Topology of a neural network.

Methods:

+ drawGraphic(neuralNetwork : NeuralNetwork) : Image
This method draws a topology of an given neural network (the neuralNetwork parameter) using the drawRectanges(), drawText() and drawLines() methods.

4.6 Class Description: DetectionResultRenderer

DetectionResultRenderer extends ClassificationResultRenderer

Methods:

+ «constructor» DetectionResultRenderer()
This is the default constructor.

+ drawGraphic(result : DetectionResult) : Image
This method draws a bounding Box on an already classified image using the drawRectangles() method.

- combineBoundingBoxProbabilities(boundingBoxes : BoundingBox[]) : Probability[]
Every image can have multiple bounding boxes. We want to show all bounding boxes but only the combined probabilities of all image classes in all bounding boxes. Therefore we combine the probabilities of all found bounding boxes and then normalize the values and return them as an array of probabilities.

5 Package Description: ControlModule

This module contains the program logic with event handlers to react and inferencers to collect all the needed information. In the following diagram you can see what needs to happen for a full image classification process and what needs to happen to clarify each task of the classes.

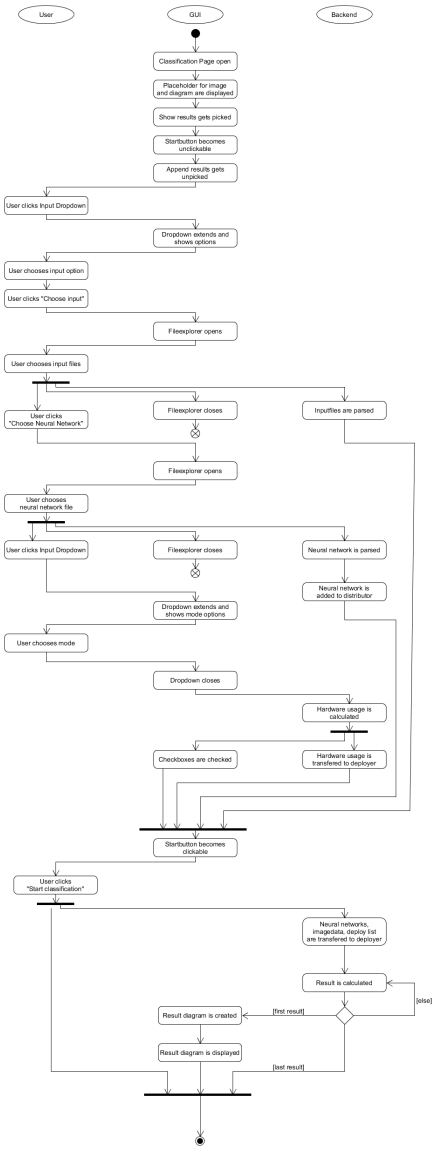
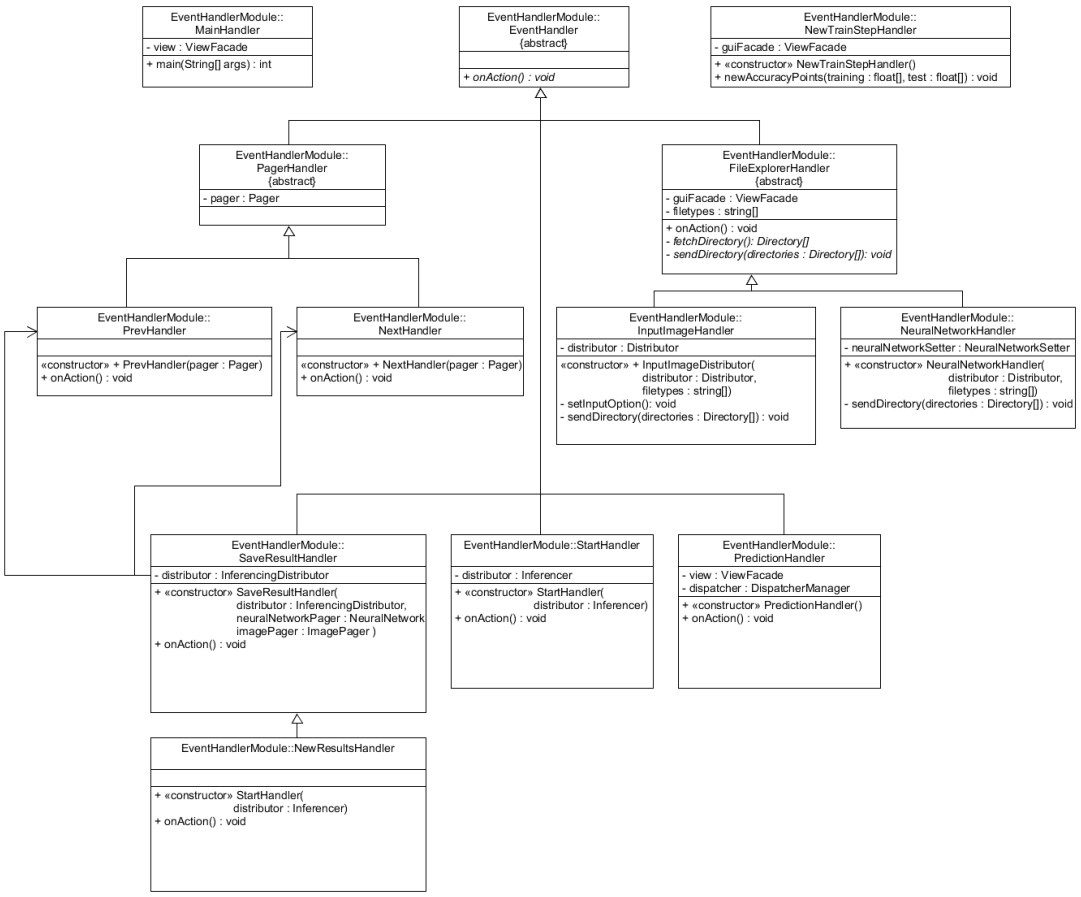


Figure 4: Activity diagram for the classification process, starts when the user opens the page and ends if the result is shown

6 Package Description: EventHandlerModule

This module contains all the event handlers of the program.
Those should be called if a certain event happens and those specify the reaction to this event.



6.1 Class Description: MainHandler

Methods:

+ main() : int
Starts the software. Creates an instance of the control facade. Upon creating this class all needed classes will be created by the constructors. This process can be seen in the following diagram:

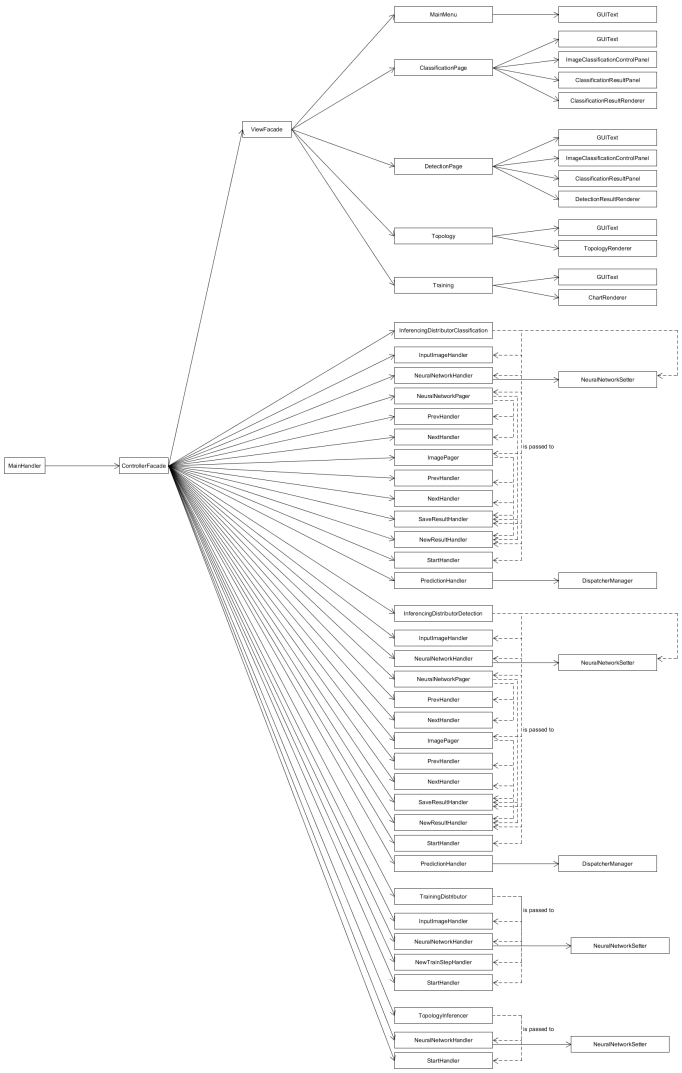


Figure 5: This diagram show what class creates which classes. Arrows show that the class creates another. Dotted arrows show that it is used in the constructor for that class.

6.2 Class Description: EventHandler

The class provides the interface for each event handler. The onActionI() - method should be called in case the event happens.

Methods:

+ onAction() : void
This method is called when the event happened. It should be implemented accordingly by the inheriting classes.

6.3 Class Description: PagerHandler

Extends EventHandler.
This class is abstract and is the base for events that happen with the pager that scrolls through results.

Attributes:

- pager : Pager
The pager that should be notified that the "Next" or "Previous" button was clicked.

6.4 Class Description: PreviousHandler

Extends PagerHandler.
The handler that is called when the "Previous" button of a diagram was clicked.

Methods:

+ «constructor» PreviousHandler(pager : Pager)
Sets its pager to the given pager.

+ onAction() : void
Calls the prev()-function of the corresponding pager.

6.5 Class Description: NextHandler

Extends PagerHandler.
The handler that is called when the "Next" button of a diagram was clicked.

Methods: + «constructor» NextHandler(pager : Pager)
Sets its pager to the given pager.
The handler that is called when the "Next" button of a diagram was clicked.

+ onAction() : void
Calls the next()-function of the corresponding pager.

6.6 Class Description: FileExplorerHandler

Extends EventHandler.
This handler is the abstract base class for the handlers that open a file explorer to get input.

Attributes:

- view : ViewFacade
The view facade that should open the FileExplorer.

- fileTypes : string[]
The file types that should be visible in the file explorer.

Methods:

+ onAction() : void
Calls the fetchDirectory() - Method and then the sendDirectory().

fetchDirectory() : string[]
Opens the fileExplorer of the view module with the fitting files visible and returns the chosen directory.

 # sendDirectory() : void
The method that should be implemented by the inhereting classes to send the directories to the correct receiver. It should then also call the startEnable()-method to check if the start button can be enabled.

6.7 Class Description: InputImageHandler

Extends FileExplorerHandler.
This handler is called when the input images should be chosen.

Attributes:

- distributor : InferencingDistributor
The distributor to send the directories to.

Methods:

«constructor» + InputImageDistributor(distributor : InferencingDistributor, filetypes : string[])
Sets class attributes to function parameters.

fetchDirectory() : string[]
Overwrites the method to first check for the input types chosen. Then the file explorer is opened.

- setInputOption() : void
The method calls the ViewFacade to set the chosen input type of the class.

sendDirectory(directories : string[]) : void
The directories are pushed to the Distributor.

6.8 Class Description: NeuralNetworkHandler

Extends FileExplorerHandler.
This handler is called when the neural network should be chosen.

Attributes:

- neuralNetworkSetter : NeuralNetworkSetter
The setter to send the directories.

Methods:

+ «constructor» NeuralNetworkHandler(distributor : Distributor, filetypes : string[])
Stores the arguments in attributes. Then creates a NeuralNetworkSetter with the distributor.

sendDirectory(directories : string[]) : void
Sends the directories to the NeuralNetworkSetter with the setNeuralNetwork(direcory : string[]) - Method.

6.9 Class Description: StartHandler

Extends EventHandler.
This handler is called when the user clicks the "Start Process" - button.

Attributes:

- distributor : Inferencer
The inferencer that should start a process.

Methods:

+ «constructor» StartHandler(inferencer : Inferencer)

+ onAction() : void
Calls the distributor to start the process.

6.10 Class Description: PredictionHandler

Extends EventHandler.
This handler is called if the user changes his platform settings leading to a new prediction of power consumption and performance.

Attributes:

- viewFacade : ViewFacade
- dispatcher : DispatcherManager
The dispatcher that runs the prediction.

Methods:

+ «constructor» PredictionHandler()
The constructor calss the ViewFacade to get an instance of the viewFacade and creates a DispatchManager.

+ onAction() : void
The viewFacade is called to get the choices for mode and hardware. This information is then sent to the dispatcher to calculate the power consumption and performance. These floats are then given to the view and the view is updated.

6.11 Class Description: SaveResultHandler

Extends EventHandler.
This handler is called if the user wants to save a result.

Attributes:

- distributor : InferencingDistributor
The distributor that should save the result.

- neuralNetworkPager : NeuralNetworkPager
The pager that stores the displayed neural network.

- imagePager : ImagePager
The pager that stores the displayed image.

Methods:

+ «constructor» SaveResultHandler(distributor : InferencingDistributor, neuralNetwork-Pager : NeuralNetworkPager imagePager : ImagePager)
Stores the arguments in attributes.

+ onAction() : void
The methods gets the current image page and current neural network page and then calls the distributor to save the result with the given ids.

6.12 Class Description: NewResultHandler

Extends EventHandler.
This handler is called if the distributor gets new results to display. It will then reset the pagers.

Methods:

+ «constructor» NewResultHandler(distributor : InferencingDistributor, neuralNetwork-Pager : NeuralNetworkPager imagePager : ImagePager)

+ onAction() : void
Gets the amount of neural networks and images and resets the pageres with it.

6.13 Class Description: NewTrainStepHandler

During the trainingprocess this handler gives the view module the information which points to draw in the training diagram.

Attributes:

- guiFacade : ViewFacade
The view module to draw the steps.

Methods:

+ «constructor» NewTrainStepHandler(view : ViewFacade)

+ newAccuracyPoints(training : float[], test : float[]) : void
Passes the given points to the viewFacade to the newTrainingFinished(training : float[], test : float[]) - method.

7 Package Description: InferencingModule

This package handles the settings and inputs of the user on a page. All this information is stored and will be used for the correct transfer of data when the user starts a process.

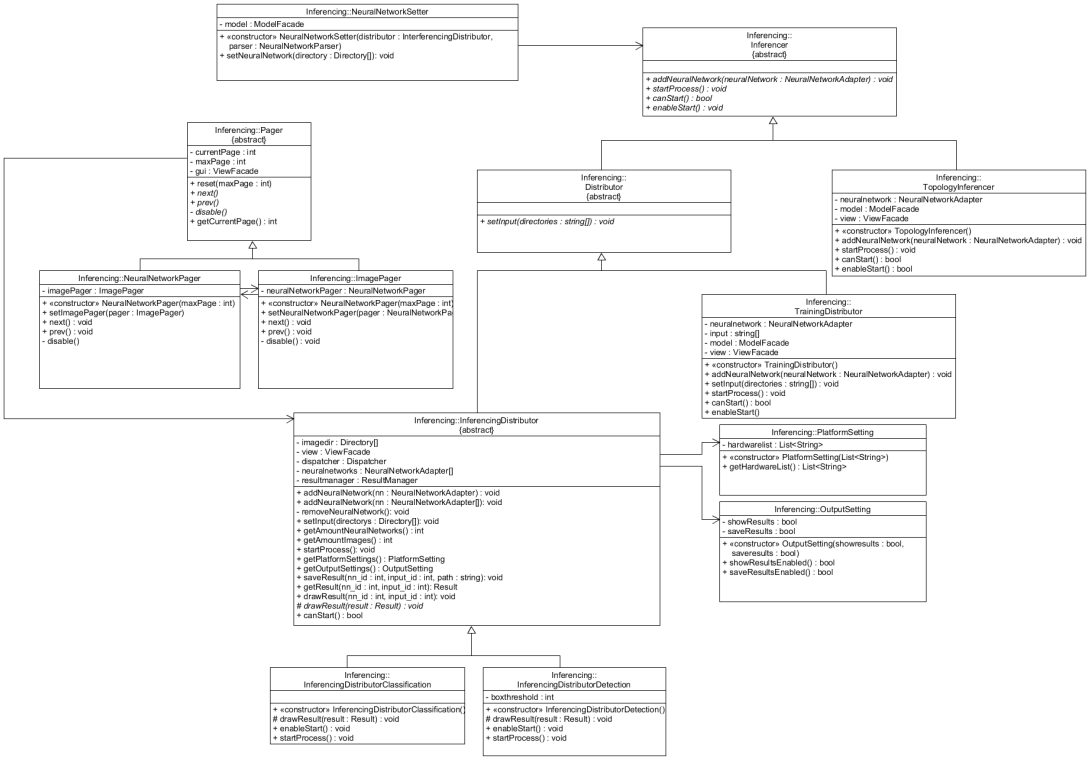


Figure 6: Package diagram for the inferencing module

7.1 Class Description: Inferencer

This class creates an interface that gives inheriting classes the method to start a process or add a neural network. This is needed in every page's task.

Methods:

+ addNeuralNetwork(neuralNetwork : NeuralNetworkAdapter) : void
The interface method to add a neural network.

+ startProcess() : void
The interface method to start a process.

+ canStart() : bool
The interface to check if a process is ready to start.

+ enableStart() : void
The interface to enable the start button.

7.2 Class Description: TopologyInferencer

Extends Inferencer.
This class handles all the information needed for the topology page.

Attributes:
- neuralnetwork : NeuralNetworkAdapter
The network to display.

- model : ModelFacade
The model facade.

- view : ViewFacade
The view to display the topology.

Methods:

+ «constructor» TopologyInferencer()
The constructor fetches the ViewFacade singleton.

+ addNeuralNetwork(neuralNetwork : NeuralNetworkAdapter) : void
A single neural network is saved.

+ startProcess() : void
Calls the viewFacade with the neural network to draw its topology.

+ enableStart() : void
Sets state of start button according to result of canStart() by giving a signal to the viewFacade.

+ canStart() : bool
Checks if a neural network is saved.

7.3 Class Description: Distributor

Extends Inferencer.
This class extends the interface of Inferencer by giving the option to set Input. This is needed by all other pages' tasks.

Methods:

+ setInput(directories : string[]) : void
The interface method to set Input.

7.4 Class Description: TrainingDistributor

Extends Distributor.
This class handles all the information needed for the training page.

Attributes:

- neuralnetwork : NeuralNetworkAdapter
The neural network to train.

- input : string[]
The input directories of the dataset.

- model : ModelFacade
- view : ViewFacade

Methods:

+ «constructor» TrainingDistributor()
The constructor fetches the ViewFacade singleton.

+ addNeuralNetwork(neuralNetwork : NeuralNetworkAdaper) : void
A single neural network is saved.

+ setInput(directories : string[]) : void
Sets the attribute input to the given inputs.

+ startProcess() : void
Starts the training process by creating a Trainer, getting the relevant data from the view and passing it onto the Trainer.

+ enableStart() : void
Sets state of start button according to result of canStart() by giving a signal to the viewFacade.

+ canStart() : bool
Checks if a neural network, and input is set.

7.5 Class Description: InferencingDistributor

Extends Distributor.
This class stores the the wanted neural networks, the directories of the image input, the mode and the chosen hardware. The class is inherited twice. Once for the image classification and once for object detection. They differ in the way that they handle the results.

Attributes:

- imageDir : string[]
The directory of the images the user chose for the process. If the process starts these directories will be given to the dispatcher.

- viewFacade : ViewFacade
The view facade.

- dispatcher : Dispatcher
The dispatcher to give the neural network, hardware list, mode and input images to when the process starts.

- neuralNetworks : NeuralNetworkAdapter[]
The neural networks that should run the process.

- resultManager : ResultManager
The distributor can get the results of an process from here.

Methods:

+ addNeuralNetwork(neuralNetwork : NeuralNetwork) : void
The distributor gets a neural network that it saves.

+ addNeuralNetwork(neuralNetwork : list<NeuralNetwork>) : void
The distributor gets multiple neural networks that it saves.

- removeNeuralNetwork() : bool
Resets the neural networks and sets them to null.

+ setInput(directories : Directory[]) : void
Sets the list of directories to use for the process. They will later be parsed and fitted to each neural network.

+ getAmountNeuralNetworks() : int
Returns number of neural networks.

+ getAmountImages() : int
Returns number of images.

+ startProcess() : void
Gives the signal to start the process. The distributor sends the collected information to the deployer that handles and distributes the tasks.

+ getPlatformSettings() : PlatformSettings
Returns the Platformsettings.

+ getOutputSettings() : PlatformSettings
Returns the Outputsettings.

+ saveResult(nn_id : int, input_id : int, path : string) : void
Gives the signal to save the specified result. The distributor finds the result and sends it to the IOModule.

+ getResult(nn_id : int, input_id : int) : void
Returns the specified result.

 # drawResult(nn_id : int, input_id : int) : void
Sends the signal to draw the specified result. The distributor finds the result. The method then calls the drawResult(result : Result) method to handle the result according to the type.

 # drawResult(result : Result) : void
This method is reponsible to send the information to the correct class. It is abstract and handled by the according inheriting class.

+ canStart() : bool
Checks if a neural network and input is saved and at least one output option is chosen.

7.6 Class Description: InferencingDistributorClassification

Extends InferencingDistributor.
The class inherits from the InferencingDistributor and implements the abstract method drawResult(result : Result) for the classification process. It is part of the pattern.

Methods:
«constructor» InferencingDistributorClassification(viewFacade : ViewFacade)

drawResult(result : Result) : void
The Result is given to the GUIFacade and the GUI is instructed to draw the results of a classification process.

+ enableStart() : void
Sets state of start button according to result of canStart() by giving a signal to the viewFacade.

+ startProcess() : void
Starts the process by creating a Dispatcher and giving it the acquired data.

7.7 Class Description: InferencingDistributorDetection

Extends InferencingDistributor.
The class inherits from the InferencingDistributor and implements the abstract method drawResult(result : Result) for the detection process. It is part of the pattern.

Attributes:

- boxthreshold : int
The maximum boxes that should be displayed.

Methods:

«constructor»InferencingDistributorDetection(viewFacade : ViewFacade)

drawResult(result : Result) : void
The Result is given to the GUIFacade and the GUI is instructed to draw the results of a detection process.

+ enableStart() : void
Sets state of start button according to result of canStart() by giving a signal to the viewFacade.

+ startProcess() : void
Starts the process by creating a Dispatcher and giving it the acquired data.

7.8 Class Description: NeuralNetworkSetter

This class gives the distributor complete neural networks.

Attributes:

- inferencer : Inferencer
The inferencer that gets the neural network.

- model : ModelFacade

Methods:

+ setNeuralNetwork(directory : Directory) : void
The setter gets the directory of the .cfg file of the wanted neural network. The directory is sent to the model facade that returns a neural network. This neural network is then given to the distributor.

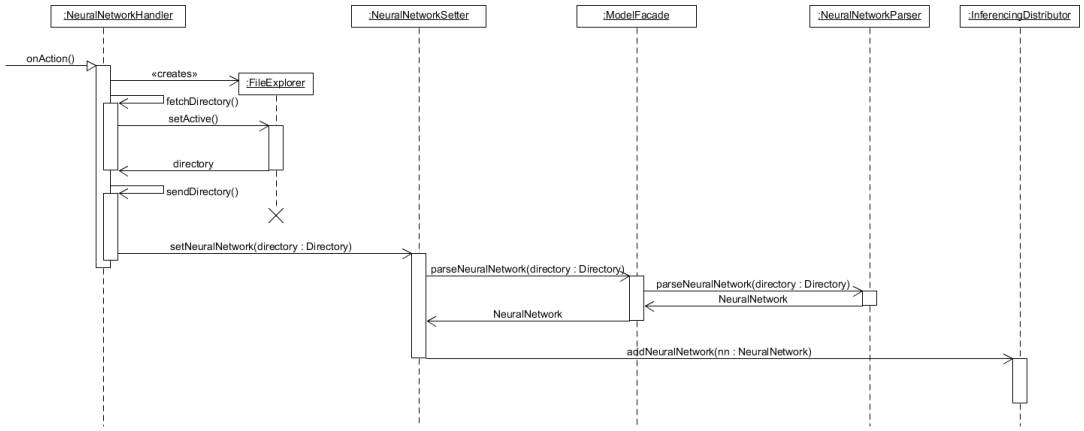


Figure 7: The process of setting up a neural network can be seen in the diagram.

7.9 Class Description: Pager

This class keeps track of the displayed results of one category.

Attributes:
- currentPage : int
Stores the number of the page it is on.

- maxPage : int
Stores the amount of pages.

- gui : ViewFacade
The view to display the pages.

Methods:

+ reset(maxPage : int)
Resets the pager to page zero and gets a new number of pages.

+ next()
The abstract method to do what happens if next button is clicked.

+ prev()
The abstract method to do what happens if prev button is clicked.

- disable()
Checks if something has to be disabled and gives the signals to the view facade.

+ getCurrentPage() : int

7.10 Class Description: NeuralNetworkPager

Extends Pager.
The pager to keep track of the neural networks.

Attributes:

- imagePager : ImagePager
The other pager to get the complete position.

Methods:

+ «constructor» NeuralNetworkPager(maxPage : int)
Fetches a viewFacade instance, sets current to zero and max to maxPage.

- disable() : void
The pager checks if the current neural network is the last or first one and disables the corresponding buttons.

+ next() : void
Increments currentPage and calls the distributor to display the new result with the new neural network. The disable()-function is then called.

+ prev() : void
Decrements currentPage and calls the distributor to display the new result with the new neural network. The disable()-function is then called.

+ setImagePager(pager : ImagePager)

7.11 Class Description: ImagePager

Extends Pager.
The pager to keep track of the images.

Attributes:

- neuralNetworkPager : NeuralNetworkPager
The other pager to get the complete page.

Methods:
+ «constructor» ImagePager(maxPage : int)
Fetches a viewFacade instance, sets current to zero and max to maxPage.

- disable() : void
The pager checks if the current image is the last or first one and disables the corresponding buttons.

+ next() : void
Increments currentPage and calls the distributor to display the new result with the new image. The disable()-function is then called.

+ prev() : void
Decrements currentPage and calls the distributor to display the new result with the new image. The disable()-function is then called.

+ setImagePager(pager : ImagePager)

7.12 Class Description: PlatformSettings

Attributes:

- hardwareList : list<string>
Stores the names of the platforms to use for the calculations.

Methods:

+ getHardwareList() : list<string>
Returns the list.

7.13 Class Description: OutputSettings

Attributes:

- showResults : bool
Defines if the results should be displayed.

- saveResults : bool
Defines if the results should be saved automatically.

Methods:

+ showResultsEnabled() : bool
Returns showResult.

+ saveResultsEnabled() : bool
Returns saveResults.

8 Package Description: NeuralNetworkAdapterModule

This package only contains the neural network adapter.

8.1 Class Description: NeuralNetworkAdapter

This class adapts the NeuralNetwork class. In our implementation all methods are the same and just passed on. In case the neural network representation changes, only this class has to be adapted and the control module can stay the same.

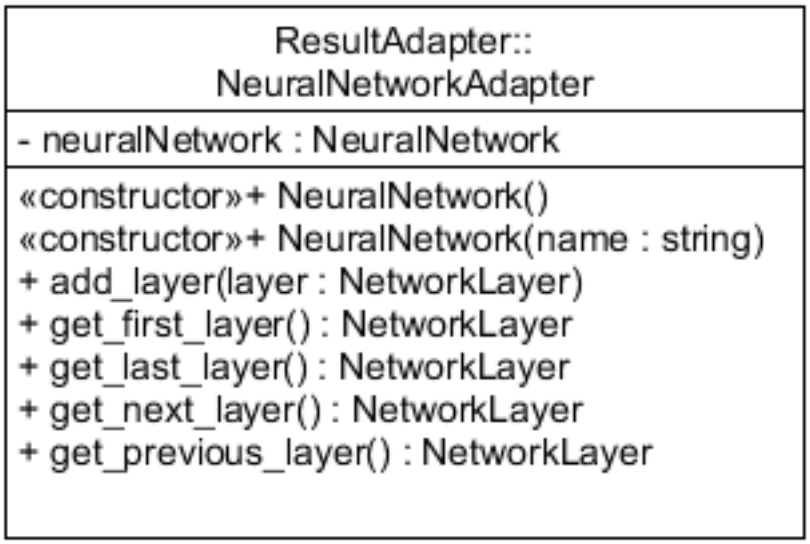


Figure 8: Package diagram for the neural network adapter

Attributes:

- neuralNetwork : NeuralNetwork
The neural network to adapt.

Methods:

All the methods just call the same methods of the neuralNetwork and return them.

«constructor»+ NeuralNetworkAdapter()
Creates a neural network.

«constructor»+ NeuralNetworkAdapter(name : string)
Creates a neural network with a given name.

+ addLayer(layer : NetworkLayer)
+ getFirstLayer() : NetworkLayer
+ getLastLayer() : NetworkLayer
+ getNextLayer() : NetworkLayer
+ getPreviousLayer() : NetworkLayer

9 Package Description: ResultAdpaterModule

The package only consists of the adapter.

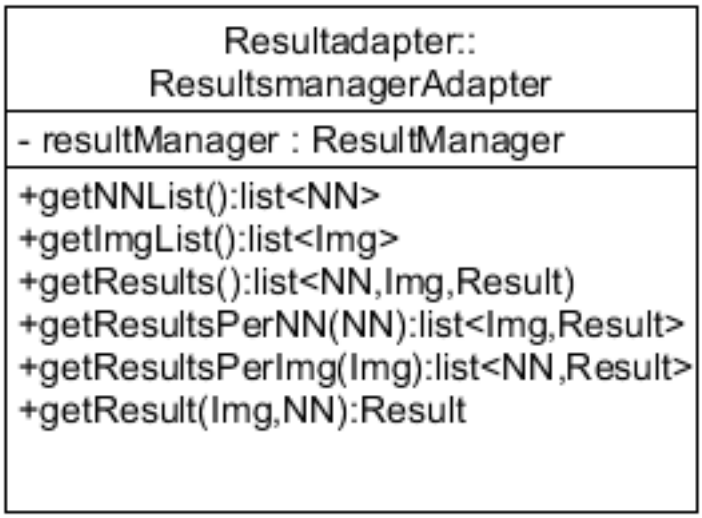


Figure 9: Package diagram of the ResultAdapterModule

9.1 Class Description: ResultAdapter

The adapter only passes the methods on to the real result class. In case the modelling of the result changes, except from this class the control module does not have to change.

Attributes:

- resultManager : ResultManager
The ResultManager to adapt.

Methods:

All the methods just call the same methods of the resultManager and return them.

+getNNList():list<NeuralNetwork>
+getImgList():list<Image>

+getResults():list<NeuralNetwork,Image,Result)
+getResultsPerNN(nn : NeuralNetwork):list<Image,Result>
+getResultsPerImg(img : Image):list<NeuralNetwork,Result>
+getResult(img : Image, nn : NeuralNetwork):Result

10 Package Description: ControlFacadeModule

This package only contains the ControlFacadeClass.

10.1 Class Description: ControlFacade

This class is the connecting class from classes outside the controller package and should be the only one called if something from outside the control package wants to interact with the control package.

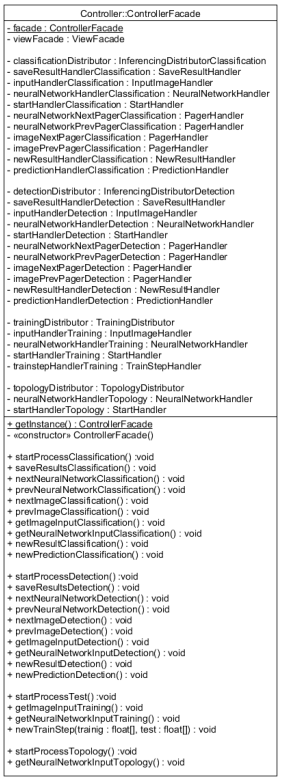


Figure 10: Package diagram of the conroller facade

11 Package Description: DispatcherModule

A package responsible for managing the execution of NeuralNetworks on multiple devices, given a list of images, a list of NeuralNetworks, a list of devices and a mode.

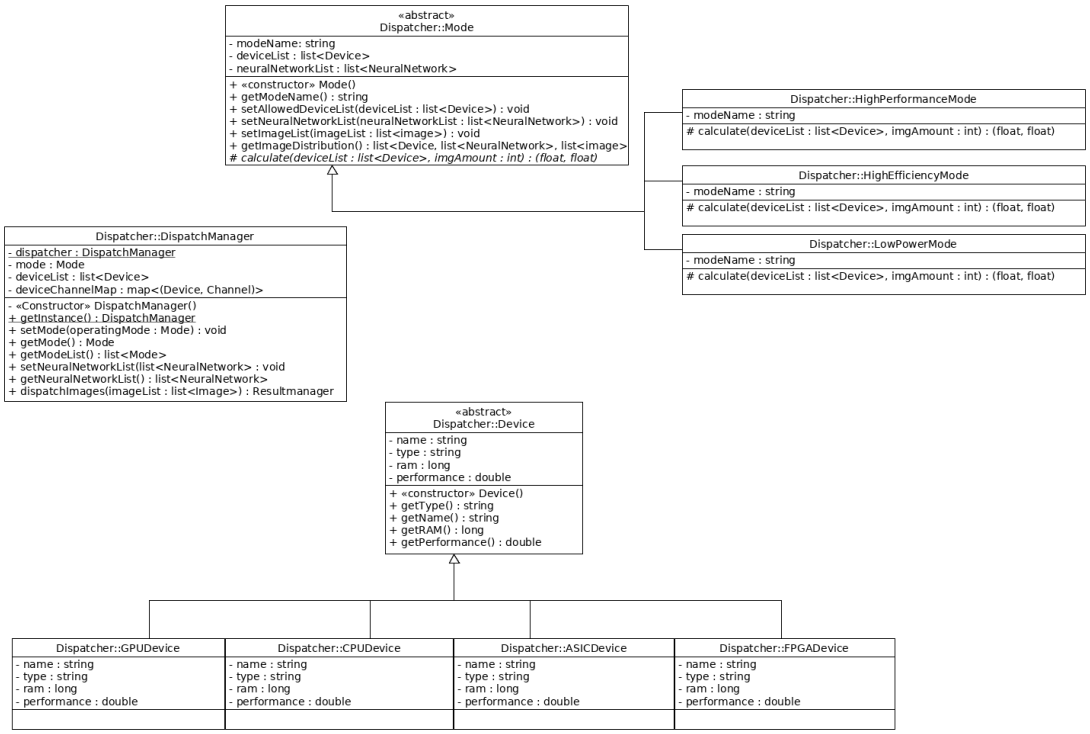


Figure 11: The dispatcher module, allows running the inference on multiple devices simultaneously.

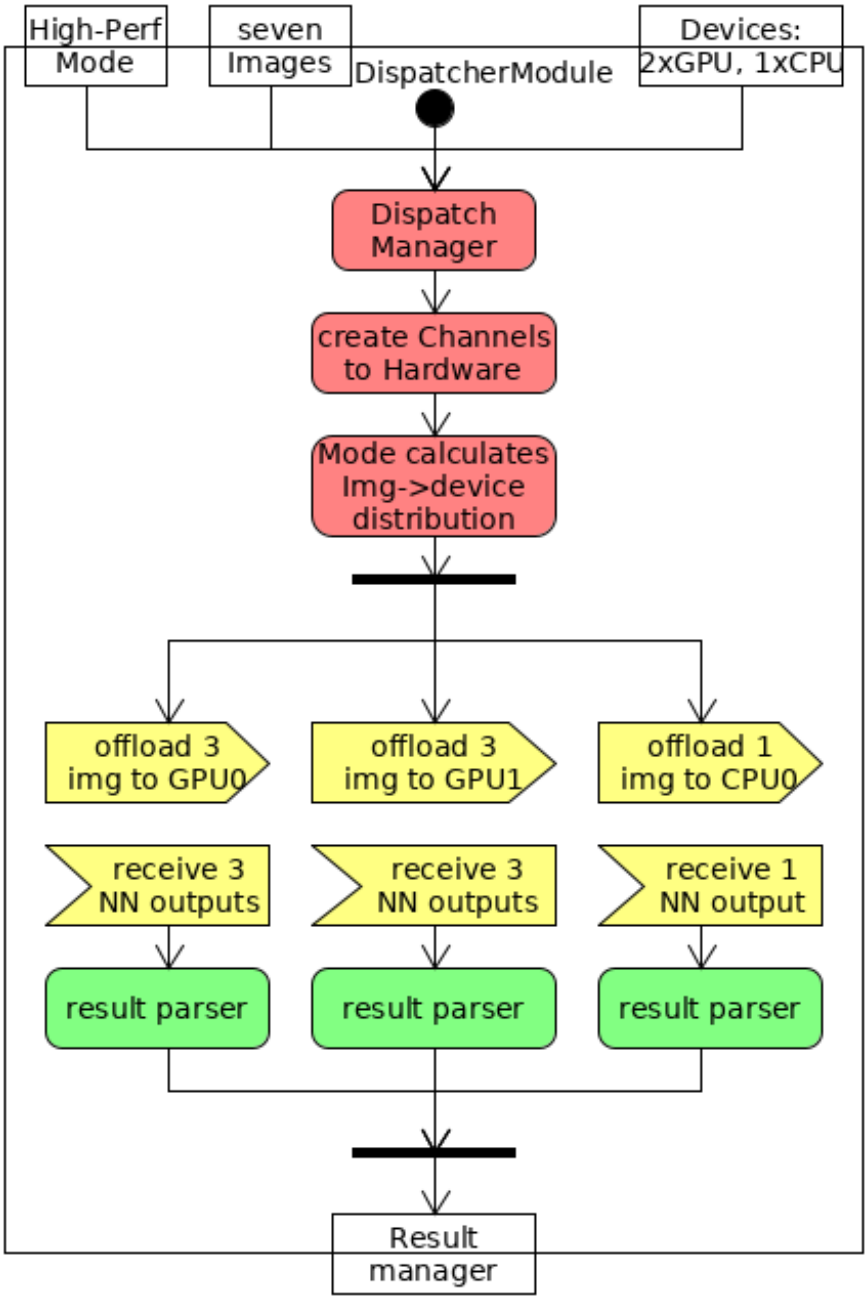


Figure 12: The process of execution the inference on multiple images with multiple devices and one neural network.

11.1 Class Description: DispatchManager

This singleton class is used to manage the execution of Neural Networks on multiple devices.

Attributes:
- dispatcher : DispatchManager

- mode : Mode
Stores the Mode selected by the user.
Default modes are high-performance, low-power and high-efficiency.

- deviceList : list<Device>
Stores the list of devices recognized by the system and allowed by the user.
The mode takes the final decision about which devices will be used.

- deviceChannelMap : map<(Device, Channel)
Stores one Channel for each device in the deviceList, allowing to communicate with that device.

Methods:

- «constructor» DispatchManager()

+ getInstance() : DispatchManager

+ setMode(operatingMode : Mode) : void
Stores the mode selected by the user.

+ getMode() : Mode
Returns the mode selected by the user.

+ getModeList() : list<Mode>
Returns a list of all available modes.

+ setNeuralNetworkList(list<NeuralNetworkAdapter> : void
Stores a list of neural networks selected by the user.

+ getNeuralNetworkList() : list<NeuralNetworkAdapter>
Returns the list of neural networks selected by the user.

+ dispatchImages(imageList : list<Image>) : Resultmanager
Runs inference on all images in the imageList, with the selected neural networks, the selected mode, and the devices allowed by the user and selected by the mode.
If multiple neural networks are selected, each network will be applied to each image.

11.2 Class Description: Mode

This class is used to determine how NeuralNetworks and Images are split across the available devices. The modes implemented from the beginning are high-performance, low-power, high-efficiency. Additional modes can be implemented by inheriting from this class.

Attributes:

- modeName: string
- deviceList : list<Device>
- neuralNetworkList : list<NeuralNetwork>

Methods:

+ «constructor» Mode()
+ getModeName() : string

+ setAllowedDeviceList(deviceList : list<Device>) : void
From all Devices connected and supported by this software,
the user can select a subset, which the software is allowed to use.

+ setNeuralNetworkList(neuralNetworkList : list<NeuralNetwork>) : void
+ setImageList(imageList : list<image>) : void

+ getImageDistribution() : list<Device, list<NeuralNetwork>, list<image>)
A simple list stating which device should receive which neural networks and which images.

 # calculate(deviceList : list<Device>, imgAmount : int) : (float, float)

11.3 Class Description: Device

This class is used to determine which information about each available device is stored. Those information can be used to determine the different modes. Necessary information are power consumption and performance.
For each hardware type, one subclass supporting this type should be created.
By default, the four hardware types: CPU, GPU, FPGA, ASIC are supported.
For each available device, one instance of the corresponding class is created.

Attributes:

- name : string
- type : string
- ram : long
- performance : double

Methods:

+ «constructor» Device()
+ getType() : string
+ getName() : string
+ getRAM() : long
+ getPerformance() : double

12 Package Description: CommunicationModule

This package is used to manage the communication between the DispatchManager and multiple Devices.

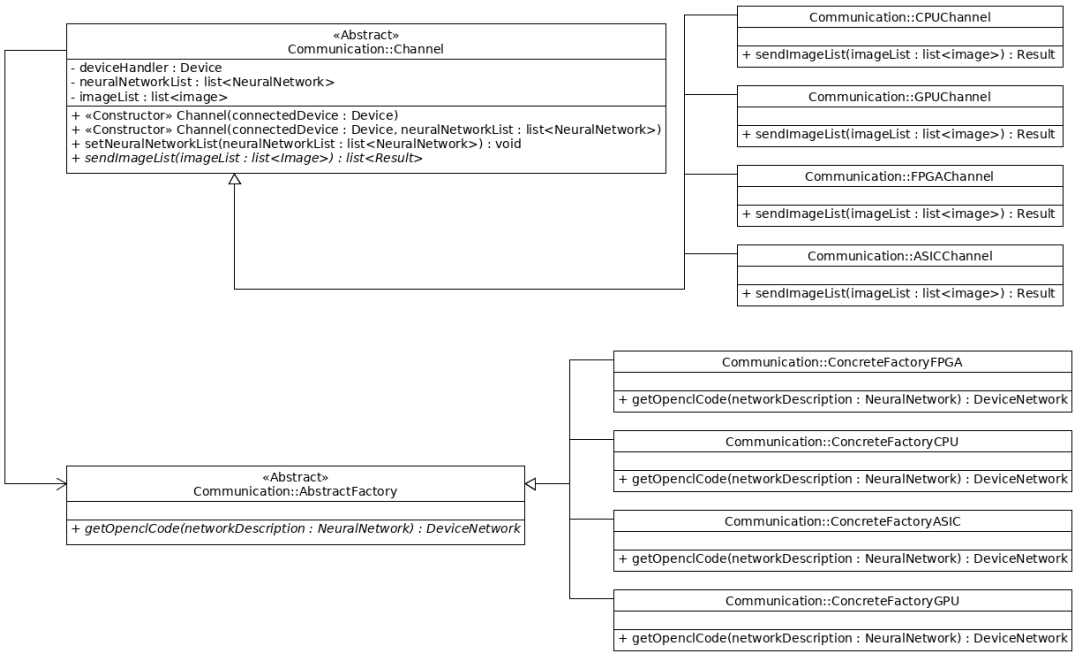


Figure 13: The communication module allows a connection between the software and the hardware.

12.1 Class Description: Channel

This class is used to manage the communication between the DispatchManager and the used devices. For each device, one Channel should be created. The class is responsible for getting the hardware specific OpenCL code based on the NeuralNetwork received, as well as calling the parser from the model to parse the network output into the result.

Attributes:

- deviceHandler : Device
The Device to which this channel is bound. Immutable.

- neuralNetworkList : list<NeuralNetworkAdapter>
A list of neural networks, which should run on this Device.

- imageList : list<image>
A list of images which should be processed on this Device.

Methods:

+ «constructor» Channel(connectedDevice : Device)
+ «constructor» Channel(connectedDevice : Device, neuralNetworkList : list<NeuralNetwork>)
+ setNeuralNetworkList(neuralNetworkList : list<NeuralNetwork>) : void
+ sendImageList(imageList : list<Image>) : ResultManager
Sends the images to the Device and runs the inference on them.
Requires at least one neural network in the neuralNetworkList.

12.2 Class Description: AbstractFactory

This class is used by the channel class, in order to get the concrete OpenCL code for the hardware, based on the given NeuralNetwork. It build's the OpenCL code layer by layer.

Methods:

+ getOpenclCode(networkDescription : NeuralNetwork) : DeviceNetwork
Returns a device-specific OpenCL implementation of the given neural network. For each Device-type a ConcreteFactory exists to build the OpenCL implementation layer-wise.
Function implemented by ConcreteFactories.

12.3 Class Description: ConcreteFactories

These classes are used by the AbstractFactory. For each supported DeviceType, one concreteFactory should be created. Each concreteFactory is responsible to build the OpenCL code for one specific Hardware Type.

Methods:

+ getOpenclCode(networkDescription : NeuralNetwork) : DeviceNetwork
Each ConcreteFactory is responsible for its specific device-type. It iterates through the given neural networks and parses it layerwise into OpenCL code.

13 Package Description: Prediction

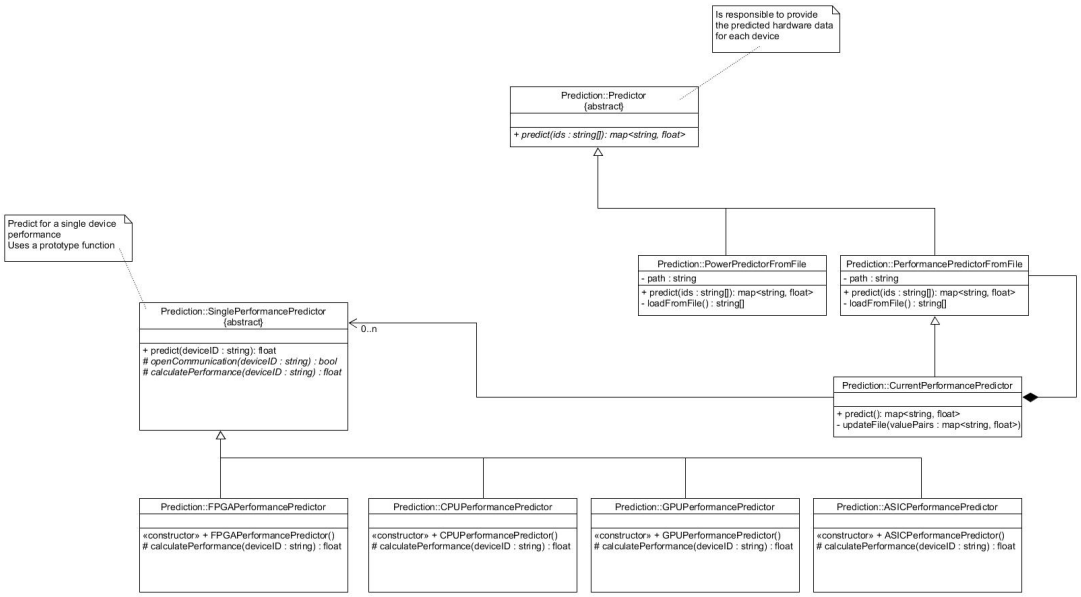


Figure 14: Package diagram for the prector module

13.1 Class Description: Power-/PerformancePredictorFromFile

Methods:

+ predict(ids : string[]): map<string, float>
Loads the values for the desired device ids which are the input from a .txt file. Returns a map of values paired to their device id.

- loadFromFile() : string[]
Loads a .txt file and returns the stored values as string array

13.2 Class Description: CurrentPerformancePredictor

Decorates the PerformancePredictorFromFile with the functionality to evaluate the current performance of the connected devices with the SinglePerformancePredictor.

Methods:

+ predict(): map<string, float>
Predicts the performance.

- updateFile(valuePairs : map<string, float>) : void
Gets the values from the predict function and then updates the file where values stored.

13.3 Class Description: SinglePerformancePredictor

Methods:

+ predict(): float
Predicts the performance of a single device with a test neural network and a test image.

- startTest(deviceID : string) : float
A pattern method.
Pattern: openCommunication –> calculatePerformance

 # openCommunication() : boolean
Opens the communication channel to the device

 # calculatePerformance(deviceID : string) : float
Starts the test and returning the calculated performance

14 Package Description: Training

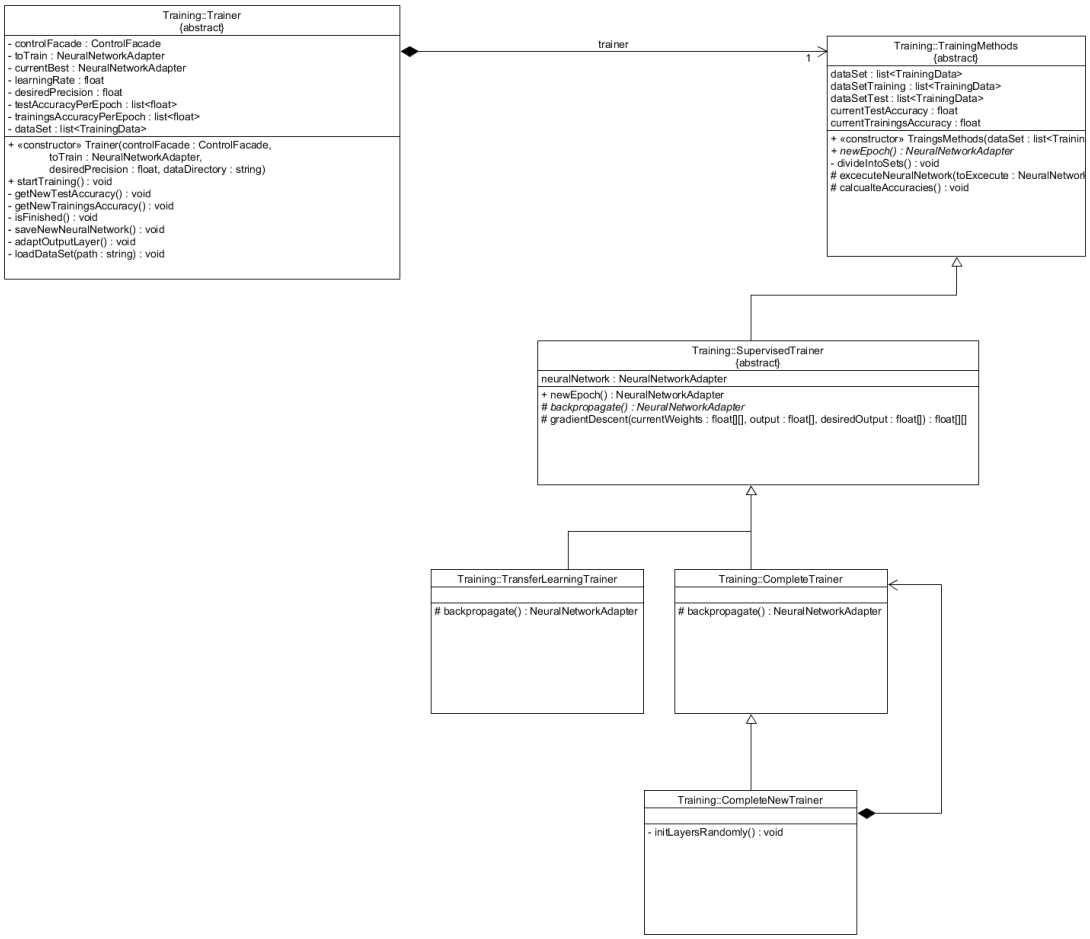


Figure 15: Package diagram of the training module

14.1 Class Description: Trainer

Manages the process of training. It decides which kind of training is used and also communicates with the controller facade to update the view.

Methods:

+ «constructor» Trainer(controlFacade : ControlFacade, toTrain : NeuralNetworkAdapter, desiredPrecision : float, dataDirectory : string)
Binds the parameters to the attributes. Furthermore, the constructor loads the datasets from the directory in data directory specified with the method loadDataSet.
Then it adapts the number of output nodes to the number of image classes.

+ startTraining() : void
Runs so many epoch as needed till the desired precision are reached. After every epoch, it checks wether it's precise enough or not, therefore it uses isFinished. If finished it calls the controller facade to finish and also saves the new trained network. In every case it gets the test accuracy and the trainings accuracy and updates the corresponding lists which then are transferred to the controller facade which then updates the progress chart with the new data point.

- getNewTestAccuracy() : void
Gets the current test accuracy from the concrete trainer and then updates the list

- getNewTrainingsAccuracy() : void
Gets the current trainings accuracy from the concrete trainer and then updates the list

- isFinished() : void
Checks if the desired precision is reached.

- saveNewNeuralNetwork() : void
Gets an instance of the file io and saves the new trained neural network.

- adaptOutputLayer() : void
Checks if the number of output nodes of the neural network corresponds with the number of image classes. If not then it adds/ removes output nodes to match the number of image classes. If new ones are added, it will add random weights.

- loadDataSet(path : string) : void
Loads the dataset for the training from the specified directory.

14.2 Class Description: TrainingMethods

Bundles the shared code for all possible kind of training. All inheritors are implementing the newEpoch method.

Methods:

+ newEpoch() : NeuralNetworkAdapter
Is called if a new trainings step begins. It organises the different steps to train the neural network. It returns the current most accurate neural network.

- divideIntoSets() : void
Divides the data set into a training set and test set.

excecuteNeuralNetwork(toExcecute : NeuralNetworkAdapter, data : TrainingData) : Result
Calls the communication moduel to excecute the neuralnetwork with the training data and returns the result of the classification/detection.

calculateAccuracies() : void
Calculates the test and training accuracy from the results.

14.3 Class Description: SupervisedTrainer

Provides the abstraction of all supervised training methods.

Methods:

+ newEpoch() : NeuralNetworkAdapter
A pattern methods which determines in which order the training step is made. For each data entry it executes the classification/ detection, then backpropagates and then calculates the acuracies.

 # backpropagate() : NeuralNetworkAdapter
Abstracts the way to backpropagate if transferlearning is used only the last connection segment is backpropagated if all layers should be trained it backpropagates through all layers.

gradientDescent(currentWeights : float[][], output : float[], desiredOutput : float[]) : float[][]
Standard implementation of gradient descent to improve the weights.

15 Package Description: ResultsModule

This package is used to represent the results from inference tasks.

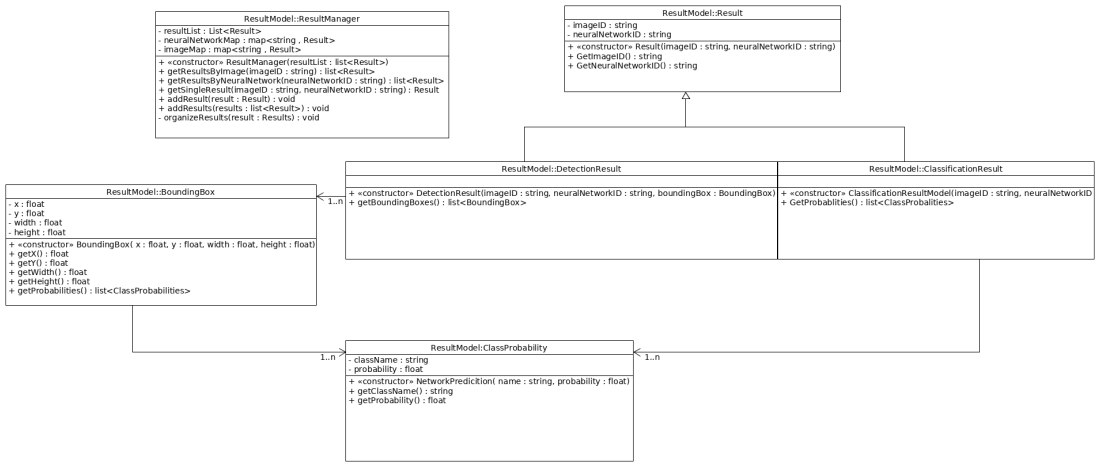


Figure 16: The results module, allowing an ordered handling of the inference results.

15.1 Class Description: ResultManager

This Class is responsible to manage the results from one inference Task. It allows to iterate results per NeuralNetwork or per Image.

Attributes:

- resultList : List<Result>
- neuralNetworkMap : map<string , Result>
- imageMap : map<string , Result>

Methods:

+ «constructor» ResultManager(resultList : list<Result>)
+ getResultsByImage(imageID : string) : list<Result>
+ getResultsByNeuralNetwork(neuralNetworkID : string) : list<Result>
+ getSingleResult(imageID : string, neuralNetworkID : string) : Result
+ addResult(result : Result) : void
+ addResults(results : list<Result>) : void
- organizeResults(result : Results) : void

15.2 Class Description: Result

This class is used to represent the outcome from one single image which run through one single NeuralNetwork. Depending on the concrete Inference Type (Classification, Detection, Segmentation,...) the corresponding subclass should be used.

Attributes:

- imageID : string
Unique identifier of the processed image.

- neuralNetworkID : string
Unique identifier of the used neural network.

Methods:

+ «constructor» Result(imageID : string, neuralNetworkID : string)

+ GetImageID() : string
Returns the unqiue image identifier.

+ GetNeuralNetworkID() : string
Returns the unique neural network identifier.

15.3 Class Description: DetectionResult

This class is used to represent the Result of one single ObjectDetection in one image by one Network.

Methods:
+ «constructor» DetectionResult(imageID : string, neuralNetworkID
: string, boundingBox : BoundingBox)

+ getBoundingBoxes() : list<BoundingBox>
Returns a list of all the BoundingBoxes created by the NeuralNetwork.

15.4 Class Description: ClassificationResult

This class is used to represent the Result of one single classification of one image by one Network.

Attributes:

Methods:

+ «constructor» ClassificationResultModel(imageID : string, neuralNetworkID : string)

+ getProbablities() : list<ClassProbalities>
Returns the list of ClassProbabilities, calculated by the neural network.

15.5 Class Description: ClassProbability

A class to group the classes, recognized by the neural network, together with the corresponding confidences of the neural network.

Attributes:

- className : string
The name of a class for which the neural network was trained.

- probability : float
The confidence of the neural network about recognizing this class.

Methods:

+ «constructor» NetworkPredicition(name : string, probability : float)

+ getClassName() : string
Returns the name of the predicted class.

+ getProbability() : float
Returns the confidence of the neural network regarding this class.

15.6 Class Description: BoundingBox

This class is used by DetectionResult.
It stores the location of one box surrounding an Object, recognized by the NeuralNetwork. It also stores the possibleClasses of this object, together with their estimated probabilities.
All positions and sizes are saved in relative values. Thus 0,0 defines the top left position, 1,1 the bottom right position.
A width of 0.4 means the box width equals 40% of the image width.

Attributes:

- x : float
- y : float
- width : float
- height : float

Methods:

+ «constructor» BoundingBox(x : float, y : float, width : float, height : float)
+ getX() : float
+ getY() : float
+ getWidth() : float
+ getHeight() : float

+ getProbabilities() : list<ClassProbabilities>
Returns the list of classes which the neural network recognized in this bounding box, together with the network confidence about these classes.

16 Package Description: ModelFacade

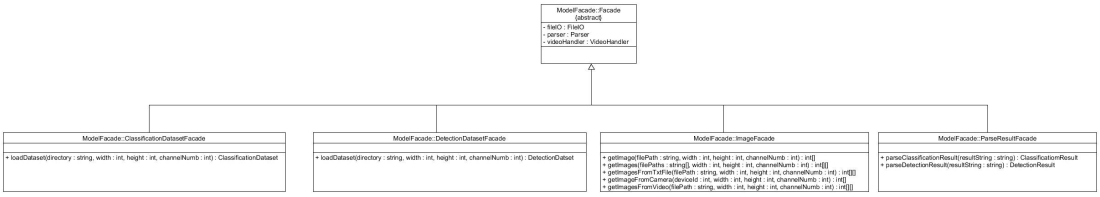


Figure 17: Package diagram for the model facade

16.1 Class Description: ClassificationDatasetFacade

The functions regarding the dataset for image classification from the model are bundled here.
Methods:
+ loadDataset(directory : string, width : int, height : int, channelNumb : int) : ClassificationDataset
Loads the dataset from the specified directory. The image names are parsed to the labels. It links the label to the image which is loaded from the directory.
It parses the image to the corrected specifications, like width, height and the channel number.
It returns the ready to use dataset.

16.2 Class Description: DetectionDatasetFacade

The functions regarding the dataset for detection from the model are bundled here.
Methods:
+ loadDataset(directory : string, width : int, height : int, channelNumb : int) : DetectionDatset
Loads the dataset from the specified directory. It looks for the correct .txt file in which are the labels for the images stored. It links the label to the image which is loaded from the directory.
It parses the image to the corrected specifications, like width, height and the channel number.
It returns the ready to use dataset.

16.3 Class Description: ImageFacade

All functions from the model facade for accessing images are bundled here.

Methods:

+ getImage(filePath : string, width : int, height : int, channelNumb : int) : int[]
Calls the fileIO and loads the image from the specified directory. Then it calls the image parser and formats the image in the correct format regarding height, width and channel number.

+ getImages(filePaths : string[], width : int, height : int, channelNumb : int) : int[][]
Calls getImage multiple times and bundles the images stored in an int array into a two dimensional int array.

+ getImagesFromTxtFile(filePath : string, width : int, height : int, channelNumb : int) : int[][]
Gets the path to a text file in which paths to images are specified and is than calling getImage for each path in the textfile.

+ getImageFromCamera(deviceId : int, width : int, height : int, channelNumb : int) : int[]
Calls the video handler to capture the next frame from the camera and parses it into the desired format.

+ getImagesFromVideo(filePath : string, width : int, height : int, channelNumb : int) : int[][] Loads a video and then cuts it into frames which are returned in the correct format.

16.4 Class Description: ParseResultFacade

It's the connection point to the result parser from the model.
Methods:
+ parseClassificationResult(resultString : string) : ClassificatiomResult
It calls the parser to create a classification result instance from a string.

+ parseDetectionResult(resultString : string) : DetectionResult
It calls the parser to create a detection result instance from a string.

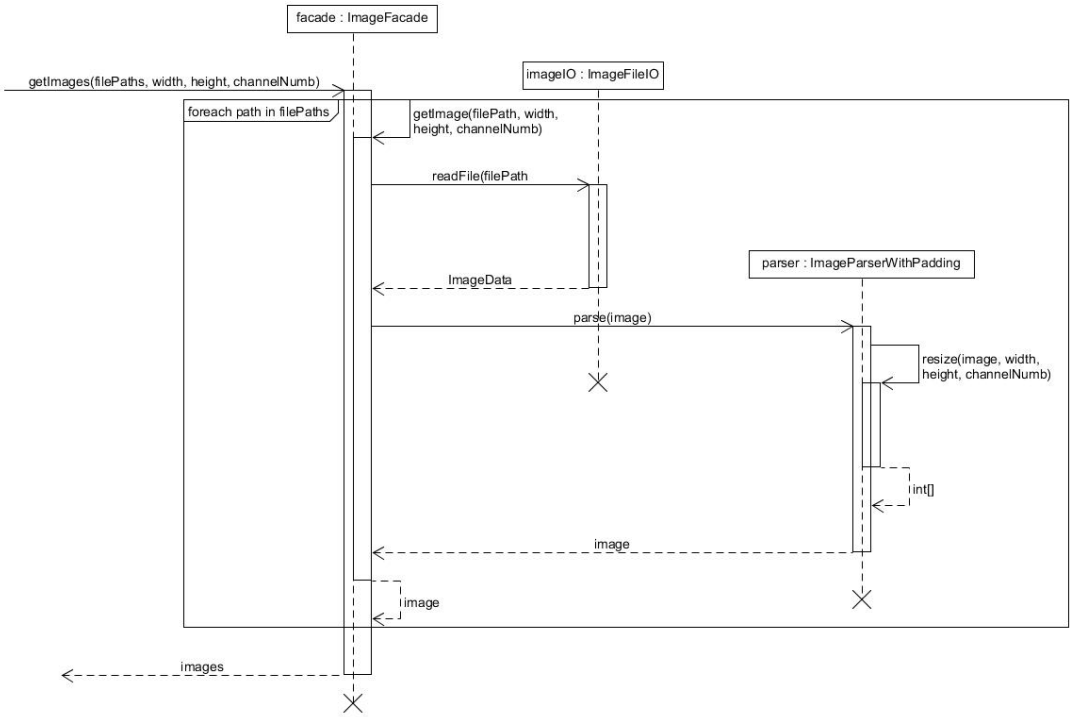


Figure 18: Sequence diagram for getting multiple images from the model

17 Package Description: IO

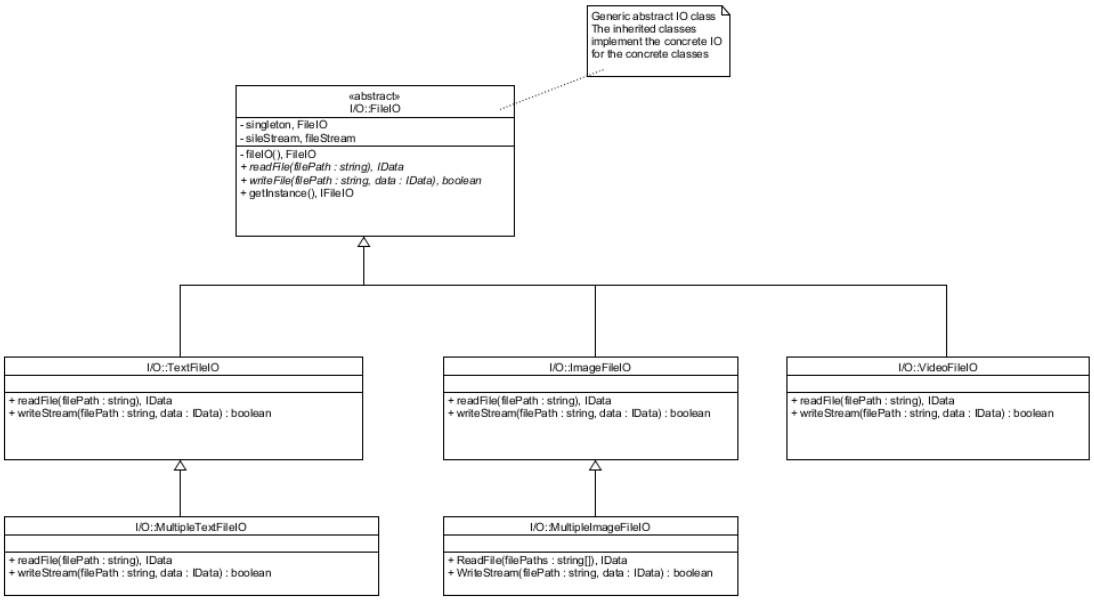


Figure 19: Package diagram for the io

17.1 Class Description: FileIO

This class abstracts the file handling. It is a singleton to avoid a double access of a file.

Methods:

+ readFile(filePath : string), Data
An abstract method for reading a file and returning the corresponding data object.

+ writeFile(filePath : string, data : Data), boolean
An abstract method for wrtiting to a file and returning the success.

17.2 Class Description: TextFileIO

Implements FileIO for loading and storing text file data for example loading and storing neural network.

17.3 Class Description: MultipleTextFileIO

Exapnds TextFileIO for supporting the loading and storing of multiple text files. It calls multiple times the base function.

17.4 Class Description: ImageIO

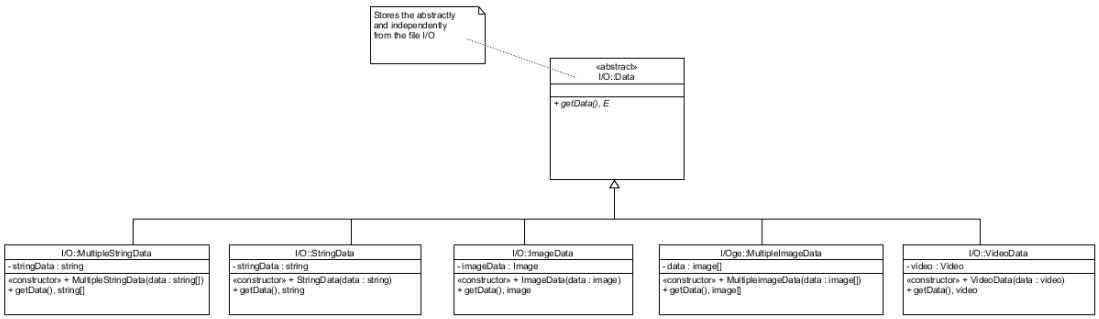
Implements FileIO for loading and storing image files.

17.5 Class Description: MultipleImageIO

Expands ImageIO for supporting the loading and storing of multiple image files. It calls multiple times the base funtion.

17.6 Class Description: VideoFileIO

Implements FileIO for loading videos.



17.7 Class Description: Data

Is the abstract class for storing data it provides a getter.

17.8 Class Description: StringData

Stores the data about a single string for example for the paths.

17.9 Class Description: MultipleStringData

Stores the data about multiple strings for example for the labels.

17.10 Class Description: ImageData

Stores the data about a single image

17.11 Class Description: MultipleImageData

Stores the data about multiple images for example for the data set.

17.12 Class Description: VideoData

Stores the data about a loaded video from the file io.

18 Package Description: Parser

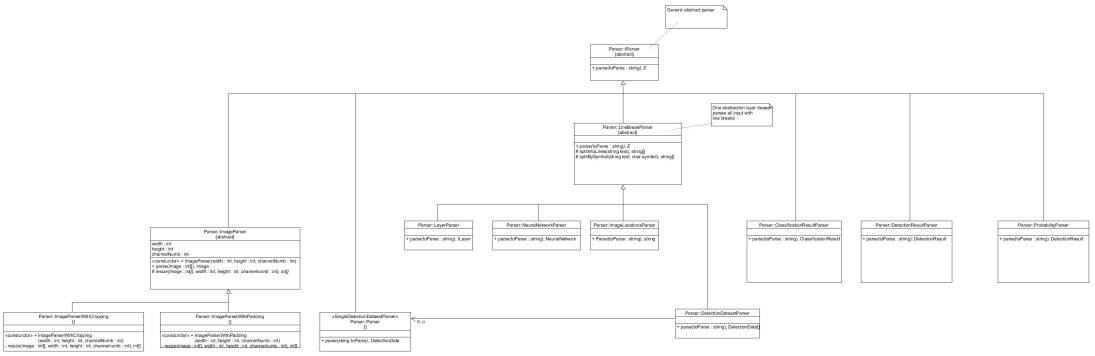


Figure 20: Package diagram for the parser

18.1 Class Description: Parser

The abstract class from which all parser are derived. Provides the abstract method parse and returning a generic which is determined in the derived classes.

18.2 Class Description: LineBreakParser

Is responsible to parse every string with linebreaks.

Methods:

+ parse(toParse : string) : E
A pattern method which first splits the text and then calls the parsing method for a string array.

parse(splittedText : string[]) : E
Parses the splitted text into the desired object like a neural network.
splitIntoLines(text : string) : string[]
Splits the whole text into a string array by the line breaks, CRLF.

splitBySymbol(string text, char symbol) . string[]
Splits a string by a specified symbol, is used by splitIntoLines with CRLF as symbol.

18.3 Class Description: NeuralNetworkParser

Parses the .cfg file into a ready to use neural network.
It uses the LayerParser and the neural network factory.

18.4 Class Description: LayerParser

Parses the the single layers into the ready to use layers with the different layer factories.

18.5 Class Description: DetectionDatasetParser

Parses a string file and the images into one large detection dataset by using the single detection data parser.

18.6 Class Description: DetectionDataParse

Parses a single string into one detection data instance.

18.7 Class Description: ImageParser

Parses an image into the desired size. Therefore it has the resize function which is called by the pattern method parse.

18.8 Class Description: ImageParserWithCropping

Resizes the image to the desired size by deleting sections from the image.

18.9 Class Description: ImageParserWithPadding

Resizes the image to the desired size by adding blank spaces around the image.

18.10 Class Description: Factories

Figure 21: Package diagram for the factories in the parser package

18.11 Class Description: LayerFactory

All inheritors are responsible to build the different layers. Therefore they all have setter methods to set the needed attributes. The setters are returning the own factory, so that following kind of code is produceable:
new LayerFactory().setA(a).setB(b).setC(c).build()
The build function then calls the constructor of the layer and returns the built layer.

18.12 Class Description: NeuralNetworkFactory

Builds the neural network from a list of layers.

18.13 Class Description: BoundingBoxFactory

Builds a new bounding box with the image class and the probabilities for the object in the bounding box as well as the position of the bounding box.

19 Package Description: VideoHandler

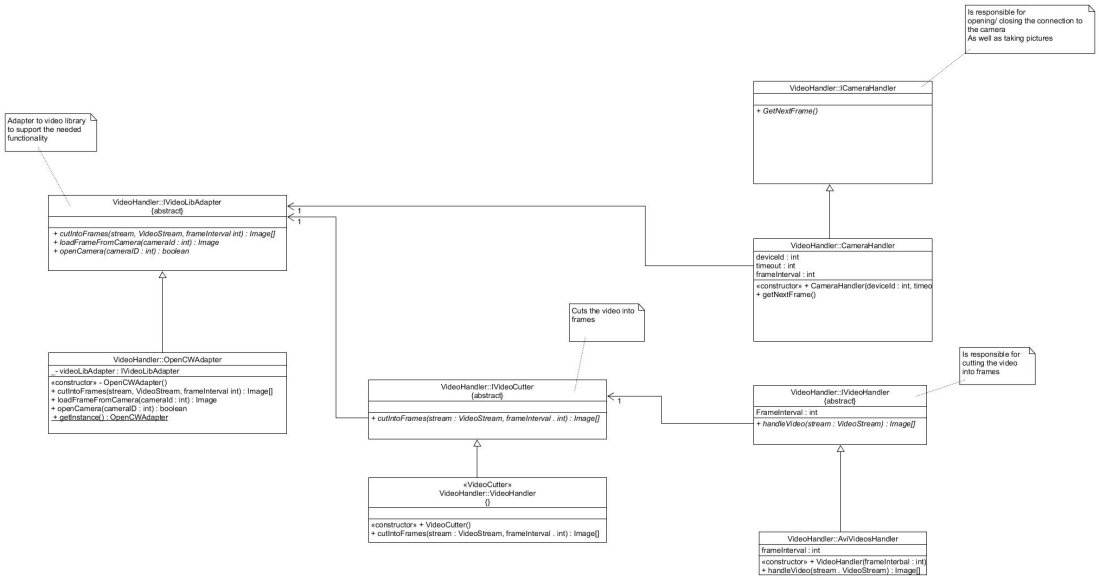


Figure 22: Package diagram of the video handler

19.1 Class Description: AviVideoHandler

This class is responsible to apply operations on .avi videos.

Methods:

«constructor» + AviVideoHandler(frameInterval : int)
Initialises an instance with an frame interval.

+ handleVideo(stream : VideoStream) : image[]
Gets an video stream as input and are applying the operations on the video. In our case the video is cut into single images. The images are single frames every frame interval. The cut images are returned.

19.2 Class Description: CameraHandler

This class is responsible for handling the camera as input source.

Methods:

«constructor» + CameraHandler(deviceId : int, timeout : int)
Initialises a connection to the camera with the specified id if connected. If the camera isn't responding the connection is after the timeout closed.

+ getNextFrame() : image
Captures the next frame the camera and returns it as image to the caller.

19.3 Class Description: VideoCutter

Methods:

«constructor» + VideoCutter()
Initialises the video cutter.

+ cutIntoFrames(stream : VideoStream, frameInterval : int) : image[]
Cuts the video into the desired frames.

19.4 Class Description: OpenCVAdapter

We are using an external library for handling the videos and cameras. This adapter is encapsulates the connection to library.
It is used as an singleton. so that for instance not two connection to the camera are coming into a race condition.

Methods:

«constructor» - OpenCVAdapter()
Opens the connection to the library OpenCV.

+ cutIntoFrames(stream : VideoStream, frameInterval : int) : image[]
Calls the corresponding function in the library and returns chosen images from the the cut video.

+ loadFrameFromCamera(cameraId : int) : Image
Calls the function from the library to capture a single frame from the connected camera.

+ openCamera(cameraID : int) : bool
Starts the connection to the camera, returns true if successful, false otherwise.

+ getInstance() : OpenCWAdapter
Manages that the adapter is a singleton.

20 Package Description: NeuralNetworkPkg

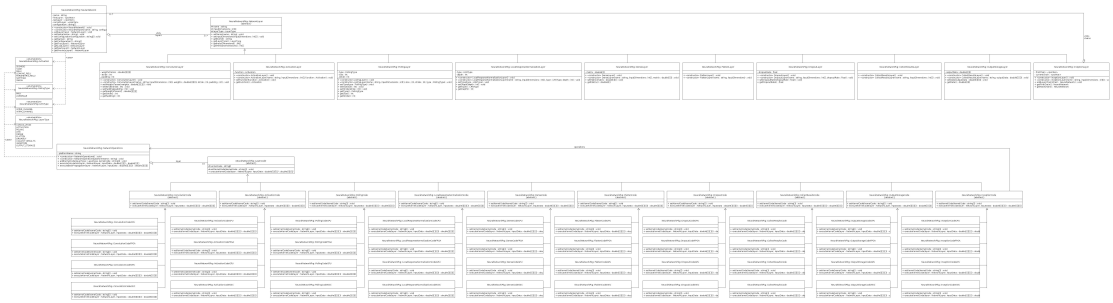


Figure 23: Package diagram of the neural network

20.1 Class Description: NeuralNetwork

Represents a neural network storing its weight tensors and dimensions/operations in an ordered list of layers.
Provides the means to add layers and get access to the data stored in them by iterating trough them.

Attributes:

- name : string
Stores the name of the neural network.

- layers : list<NetworkLayer>
Stores a list of neural network layers of different kind

- firstLayer : <pointer>
Stores the a pointer to the first layer of the neural network.

- lastLayer : <pointer>
Stores the a pointer to the last layer of the neural network.

- currentLayer : <pointer>
Stores the a pointer to the current layer of the neural network.

- configuration : string[]
The output configuration of the neural network.

Methods:

+ «constructor» NeuralNetwork() : void
Constructs an empty neural network with a default name.
No return value.

+ «constructor» NeuralNetwork(name : string, configuration : string[]) : void
Constructs an empty neural network with a given name.
Inputs:
-> name : string - The name that should be set to the newly constructed neural network.
-> configuration : string[] - The output configuration for the neural network.
No return value.

+ addLayer(layer : NetworkLayer) : void
Adds a layer to the network right after the one pointed to by the lastLayer pointer and updates the pointer to point to it. In case the neural network was empty prior to adding a layer, the firstLayer pointer also gets updated to point to the newly added layer.
Inputs:
-> layer : NetworkLayer - Layer to be added to the neural network.
No return value.

+ setName(name : string) : void
Sets the name of the neural network.
Inputs:
-> name : string - The name that should be set to the neural network.
No return value.

+ setConfiguration(configuration : string[]) :void
Sets the configuration of the neural network.
Inputs:
-> configuration : string[] - The output configuration for the neural network.
No return value.

+ getName() : string
Returns the name of the neural network.

+ getConfiguration() : string[]
Returns the configuration of the neural network.

+ getFirstLayer() : NetworkLayer
Returns the first layer of the neural network and sets the currentLayer pointer to the first layer.
Returns the first layer or NULL in case the neural network is empty.

+ getLastLayer() : NetworkLayer
Returns the last layer of the neural network and sets the currentLayer pointer to the last layer.
Returns the last layer or NULL in case the neural network is empty.

+ getNextLayer() : NetworkLayer
Returns the layer of the neural network after the one the currentLayer pointer is set to and sets the currentLayer pointer to that layer.
Returns the next layer or NULL in case the network is empty or the currentLayer pointer was on the last layer prior to calling the function.

+ getPreviousLayer() : NetworkLayer
Returns the layer of the neural network before the one the currentLayer pointer is set to and sets the currentLayer pointer to that layer.
Returns the previous layer or NULL in case the network is empty or the currentLayer pointer was on the first layer prior to calling the function.

20.2 Class Description: NetworkLayer

Represents a standard wrapper for a single layer holding its name, input dimensions and type.

Attributes:

name : string
Stores the name of the neural network layer.

inputDimensions : int[]
Stores the dimensions of the input tensor for the layer (useful for checking if needed and for the topology representation).

layerType : LayerType
Stores the type of the layer in the form of an enumeration.

Methods:

+ setName(name : string) : void
Sets the name of the neural network layer (very optional).
Inputs:
-> name : string - The name that should be set to the neural networklayer.
No return value.

+ setInputDimensions(inputDimensions : int[]) : void
Sets the input dimensions of the neural network layer.
Inputs:
-> inputDimensions : int[] - An integer vector storing width, heigth and depth.
No return value.

+ getName() : string
Returns the name of the neural network layer.

+ getLayerType() : LayerType
Gets the type of the neural network layer.
Returns a LayerType.

+ getInputDimensions() : int[]
Gets the input dimensions of the neural network layer.
Returns an integer vector storing width, heigth and depth.

+ getTensorDimenssions() : int[]
Sets up an interface to be used by the deriving clases to provide information about the structure of their tensor if they have any. Returns an integer vector.

20.3 Class Description: ConvolutionLayer

Represents a single convolutional layer with its weights tensor, stride and padding options.
Provides the means to construct it and get access to the data stored inside.

Attributes:

- weigthsTensor : double[][][][][]
Stores the tensor with all the weigths needed for the convolution.

- stride : int
Stores the stride with wich the convolution is performed.

- padding : int
Stores the padding used for the convolution.

Methods:

+ «constructor» ConvolutionLayer() : void
Constructs an empty convolutional layer.
No return value.

+ «constructor» ConvolutionLayer(name : string, inputDimensions : int[], weights : double[][][][][], stride : int, padding : int) : void
Constructs a fully initialised convolutional layer.
Inputs:
-> name : string - Name to be added to the layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
-> weights : double[][][][][] - Weigths needed for the convolution.
-> stride : int - Stride with wich the convolution is performed.
-> padding : int - Padding used for the convolution.
No return value.

+ setWeigthsTensor(weights : double[][][][][][]) : void
Sets the weigths needed for the convolution.
Inputs:
-> weights : double[][][][][][] - Weigths needed for the convolution.
No return value.

+ setStride(stride : int) : void
Sets the stride with wich the convolution is performed.
Inputs:
-> stride : int - Stride with wich the convolution is performed.
No return value.

+ setPadding(padding : int) : void
Sets the padding used for the convolution.
Inputs:
-> padding : int - Padding used for the convolution.
No return value.

+ getWeigthsTensor() : double[][][][][][]
Returns the weigths tensor for the convolution.

+ getStride() : int
Returns the stride with wich the convolution is performed.

+ getPadding() : int
Returns the padding used for the convolution.

20.4 Class Description: ActivationLayer

Represents a single activation layer its activation function.
Provides the means to construct it and get access to the data stored inside.

Attributes:
- function : Activation
Stores information about the activation function to be used in the form of an enumeration.

Methods:
+ «constructor» ActivationLayer() : void
Constructs an empty activation layer.
No return value.

+ «constructor» ActivationLayer(name : string, inputDimensions : int[], function : Activation) : void
Constructs a fully initialised convolutional layer.
Inputs:
-> name : string - Name of the neural network layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
-> function : Activation - Information about the activation function to be used.
No return value.

+ setFunction(function : Activation) : void
Sets the activation function to be used.
Inputs:
-> function : Activation - Information about the activation function to be used.
No return value.

+ getFunction() : Activation
Returns information about the activation function to be used.

20.5 Class Description: PollingLayer

Represents a single polling layer its polling type, size and stride.
Provides the means to construct it and get access to the data stored inside.

Attributes:
- type : PollingType
Stores information about the type of the polling in the form of an enumeration.

- size : int
Stores the size with wich the polling is performed (ex: 3 means each 3x3 block gets polled).

- stride : int
Stores the stride with wich the polling is performed.

Methods:
+ «constructor» polingLayer() : void
Constructs an empty polling layer.
No return value.

+ «constructor» polingLayer(name : string, inputDimensions : int[], size : int, stride : int, type : PollingType) : void
Constructs a fully initialised a polling layer.
Inputs:
-> name : string - Name to be added to the layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
-> size : int - With wich the polling is performed.
-> stride : int - Stride with wich the convolution is performed.
-> type : PollingType - Information about the type of the polling.

+ setType(type : PollingType) : void
Sets the type of the polling.
Inputs:
-> type : PollingType - Information about the type of the polling.
No return value.

+ setSize(size : int) : void
Sets the size with wich the polling is performed.
Inputs:
-> size : int - Size with wich the polling is performed.
No return value.

+ setStride(stride : int) : void
Sets the stride with wich the polling is performed.
Inputs:
-> stride : int - Stride with wich the polling is performed.
No return value.

+ getType() : PollingType
Returns information about the type of the polling in the form of an enumeration.

+ getSize() : int
Returns the strade with wich the convolution is performed.

+ getStride() : int
Returns the strade with wich the convolution is performed.

20.6 Class Description: LocalResponseNormalizationLayer

Represents a single local response normalisation with its local response normalisation type and depth.
Provides the means to construct it and get access to the data stored inside.

Attributes:

- type : LRNType
Stores information about the type of the normalisation in the form of an enumeration.

- depth : int
Stores the depth in wich the normalisation is performed.

Methods:

+ «constructor» LocalResponseNormalizationLayer() : void
Constructs an empty local linear normalisation layer.
No return value.

+ «constructor» LocalResponseNormalizationLayer(name : string, inputDimensions : int[], type : LRNType, depth : int) : void
Constructs a fully initialised local linear normalisation layer.
Inputs:
-> name : string - Name to be added to the layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
-> type : LRNType - Information about the type of the normalisation.
-> depth : int - Depth in wich the normalisation is performed.
No return value.

+ setType(type : LRNType) : void
Sets the type of the normalisation.
Inputs:
-> type : PollingType - Information about the type of the normalisation.
No return value.

+ setDepth(depth : int) : void
Sets the stride with wich the polling is performed.
Inputs:
-> stride : int - Stride with wich the polling is performed.
No return value.

+ getType() : LRNType
Returns information about the type of the normalisation in the form of an enumeration.

+ getDepth() : int
Returns the depth in wich the normalisation is performed.

20.7 Class Description: DenseLayer

Represents a single dense (fully connected) layer with its local response normalisation type and depth.
Provides the means to construct it and get access to the data stored inside.

Attributes:

- matrix : double[][]
Stores the matrix with all the weigths needed for the fully connected calculations.

Methods:

+ «constructor» DenseLayer() : void
Constructs an empty dense layer.
No return value.

+ «constructor» DenseLayer(name : string, inputDimensions : int[], matrix : double[][]) : void
Constructs a fully initialised dense layer.
Inputs:
-> name : string - Name to be added to the layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
-> matrix : double[][] - Matrix for the fully connected calculations.
No return value.

+ setMatrix(matrix : double[][]) : void
Sets the matrix for the fully connected calculations.
Inputs:
-> matrix : double[][] - Matrix for the fully connected calculations.
No return value.

+ getMatrix() : double[][]
Returns the matrix for the fully connected calculations.

20.8 Class Description: FlattenLayer

Represents a single flattening layer, a dummy layer holding no information other than its existence telling the program to rearrange the 3D working data tensor into a vector. Useful before dense layers.

Attributes:

Methods:

\+ «constructor» FlattenLayer() : void
Constructs an empty flattening layer.
No return value.

\+ «constructor» FlattenLayer(name : string, inputDimensions) : void
Constructs a fully initialised flattening layer.
Inputs:
-> name : string - Name to be added to the layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
No return value.

20.9 Class Description: DropoutLayer

Represents a single dropout layer, that randomly selects bits of the working data not to pass along to the next layer based on a given probability.
Provides the means to construct it and get access to the data stored inside.

Attributes:

\- dropoutRate : float
Stores the probability for a neuron to be droped by the dropout layer.

Methods:

+ «constructor» DropoutLayer() : void
Constructs an empty dropout layer.
No return value.

+ «constructor» DropoutLayer(name : string, inputDimensions : int[], dropoutRate : float) : void
Constructs a fully initialised dropout layer.
Inputs:
-> name : string - Name to be added to the layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
-> dropoutRate : float - The dropout rate for the layer.
No return value.

+ setDropoutRate(dropoutRate : float) : void
Sets the dropout rate for the neural network layer.
Inputs:
-> dropoutRate : float - The dropout rate for the layer.
No return value.

+ getDropoutRate() : float
Returns the dropout rate of the layer.

20.10 Class Description: CollectResultsLayer

Represents a single collect results layer, a dummy layer holding no information other than its existence telling the program this is a place to check the output of the network. Can be used more than once in a neural network (ex. GoogleNet).

Attributes:

Methods:

+ «constructor» CollectResultsLayer() : void
Constructs an empty convolutional layer.
No return value.

+ «constructor» CollectResultsLayer(name : string, inputDimensions : int[]) : void
Constructs a fully initialised collect results layer.
Inputs:
-> name : string - Name to be added to the layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
No return value.

20.11 Class Description: OutputStorageLayer

Represents a single output storage layer, has no meaning for the neural network as a whole, but provides a cheap structure to store the working data 3D tensors (layer outputs) in a structure mimmicking that of the neural network. Usefull when the data from every step needs to be stored (ex. for training).

Attributes:
- outputData : double[][][][]
Stores data (3D Tensor).

Methods:
+ «constructor» CollectResultsLayer() : void
Constructs an empty output storage layer.
No return value.

+ «constructor» CollectResultsLayer(name : string, outputData : double[][][][]) : void
Constructs a fully initialised collect results layer.
Inputs:
-> name : string - Name to be added to the layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
-> outputData : double[][][][] - Data to be stored.
No return value.

+ setData(outputData : double[][][][]) : void Sets the data to be stored.
Inputs:
-> outputData : double[][][][] - Data to be stored.
No return value.

+ getData() : double[][][][] Returns the stored data.

20.12 Class Description: InceptionLayer

Represents a single inception layer, holding its own chains of layers (implemented here as a list of neural networks), the results of wich later get concatenated together.
Provides the means to construct it and get access to the data stored inside.

Attributes:
- firstChain : <pointer>
Stores the pointer to the first chain of the inception layer.

- currentChain : <pointer>
Stores the pointer to the current chain of the inception layer.

- chains : list<NeuralNetwork>
Stores a list of different chains to be performed with the same input data in the inception layer.
The output of these chains gets concatenated at the end.

Methods:
+ «constructor» InceptionLayer() : void
Constructs an empty inception layer.
No return value.

+ «constructor» InceptionLayer(name : string, inputDimensions : int[]) : void
Constructs an initialised inception layer.
Inputs:
-> name : string - Name to be added to the layer.
-> inputDimensions : int[] - Dimensions of the input tensor for the layer.
No return value.

+ addLayerChain(chain : NeuralNetwork) : void
Adds a new chain to the chain list of the inception layer.
Inputs:
-> chain : NeuralNetwork - Chain to be added.
No return value.

+ getFirstChain() : NeuralNetwork
Gives the first chain of the inception layer and sets the currentLayer pointer to the first layer.
Returns a NeuralNetwork object or NULL in case the chains list is empty.

+ getNextChain() : NeuralNetwork
Gives the chain of the inception layer after the one the currentLayer pointer is set to and sets the currentLayer pointer to that layer.
Returns a NeuralNetwork object or NULL n case the chain list is empty or the current-Layer pointer was on the last chain prior to calling the function.

20.13 Class Description: NetworkOperations

Attributes:

- platformName : string
Identifies the platform covered with the current instance of the network operations.

Methods:

+ «constructor» NetworkOperations() : void
Constructs NetworkOperations instance with a default latform name.
No return value.

+ «constructor» NetworkOperations(platformName : string) : void
Constructs NetworkOperations instance.
Inputs:
-> platformName : string - Name of platform covered with the current instance of the network operations.
No return value.

+ addKernelCode(layerType : Layertype, kernelCode : string[]) : void
Sets array of openCL codes each saved in a string for a speciffic layer type.
Inputs:
-> layerType : Layertype - Information about the type of the layer in the form of an enumeration.
-> kernelCode : string[] - OpenCL functions each saved in a string.
No return value.

+ executeCalculation(layer : NetworkLayer, inputData : double[][][][]) : double[][][][]
Executes openCL functions for a neural network layer calculation and an input data by chosing which LayerCode class's executeKernelCode() funktion to invoke based on the layer given.
Inputs:
-> layer : NetworkLayer - neural network to be used.
-> inputData : double[][][][] - input data to be used.
Returns the output data after running the input data trough the neural network layer.

+ executeBackPropagation(layer : NetworkLayer, inputData : double[][][][][]) : double[][][][][]
+ executeCalculation(layer : NetworkLayer, inputData : double[][][][]) : double[][][][]
Executes openCL functions for a neural network layer backpropagation and an input data by chosing which LayerCode class's executeKernelCode() funktion to invoke based on the layer given.
Inputs:
-> layer : NetworkLayer - neural network to be used.
-> inputData : double[][][][][] - input data to be used.
Returns the output data after running the input data trough the neural network layer.

20.14 Class Description: LayerCode

Attributes:
kernelCode : string[]

Methods:
 # setKernelCode(kernelCode : string[]) : void
Sets an array of openCL functions (kernel code) saved in strings.
Inputs:
-> kernelCode : string[] - OpenCL functions each saved in a string.
No return value.

+ executeKernelCode(layer : NetworkLayer, inputData : double[][][][][][][]) : double[][][][][][][]/
Executes openCL functions for a neural network layer and an input data.
Inputs:
-> layer : NetworkLayer - neural network to be used.
-> inputData : double[][][][][][] - input data to be used.
Returns the output data after running the input data trough the neural network layer.

Classes with this interface:

ConvolutionCodeCPU
ConvolutionCodeFPGA
ConvolutionCodeGPU
ConvolutionCodeASIC
ActivationCodeASIC
ActivationCodeFPGA
ActivationCodeCPU
ActivationCodeGPU
PollingCodeASIC
PollingCodeGPU
PollingCodeFPGA
PollingCodeCPU
LocalResponseNormalizationCodeCPU
LocalResponseNormalizationCodeFPGA
LocalResponseNormalizationCodeGPU
LocalResponseNormalizationCodeASIC
FlattenCodeASIC
FlattenCodeGPU

FlattenCodeFPGA
FlattenCodeCPU
DropoutCodeASIC
DropoutCodeGPU
DropoutCodeFPGA
DropoutCodeCPU
CollectResultsCode
CollectResultsCode
CollectResultsCode
CollectResultsCode
OutputStorageCodeCPU
OutputStorageCodeASIC
OutputStorageCodeGPU
OutputStorageCodeFPGA
InceptionCodeASIC
InceptionCodeGPU
InceptionCodeFPGA
InceptionCodeCPU
DenseCode
DenseCodeASIC
DenseCodeGPU
DenseCodeFPGA
DenseCodeCPU

20.15 Enumeration Description: Activation

SIGMOID
Represents a sigmoid activation function.
TANH
Represents a hyperbolic tangent activation function.
RELU
Represents a rectified linear unit activation function.
LEAKING_RELU
Represents a leaking rectified linear unit activation function.
PARAMETRIC_RELU
Represents a parametric rectified linear unit activation function.
SOFTMAX
Represents a normalized exponential activation function.
SWISH
Represents google's swish activation function.

20.16 Enumeration Description: PollingType

MAX
Represents max polling.
AVEREAGE
Represents average polling.

20.17 Enumeration Description: LRNType

INTER_CHANNEL
Represents a flat local response normalisation.
INTRA_CHANNEL
Represents a deep local response normalisation.

20.18 Enumeration Description: LayerType

CONVOLUTION
Represents a convolutional layer.
ACTIVATION
Represents an activation layer.
POLING
Represents a polling layer.
LRN
Represents a local response normalisation layer.
DENSE
Represents a dense layer.
FLATTEN
Represents a flattening layer.
DROPOUT
Represents a dropout layer.
COLLECT_RESULTS
Represents a collect results layer.
INCEPTION
Represents an inception layer.
OUTPUT_STORAGE
Represents a output storage layer.