HePICS Design Document

December 23, 2018

Contents

1 Functionality Description

In the model part, classes realize the management of all the data and logic. They load pathes of input images and access their features. The classification results are stored in the type of string to be shown in various forms. The logic of classification is determined by the neural network being used. Its topology defines how different layers interact with each other. With the assistance of all the layers, the forward propagation functionality can be implemented, which is the core of algorithms for image classification with pre-trained model.

The view realizes the user interfaces. The welcome window and the main window, which contains three different sections, guide the user to upload his input files, select available platforms and operating mode, let him control the process and show the prediction results.

The controller accepts user inputs handled by the components of the views and converts them to the commands. The classifier controls the progress of processing and assigns the scheduler to dispatch work to workers in heterogeneous platforms. In addition, the poller checks the requests from external systems regularly. After receiving the correct request with the image path, the classifier starts the process of classification.

2 Class Diagram Description

In this diagram several design patterns were used.
In order to decouple major components and allow parallel development we use the Model-View-Controller as a global architecture. The class diagram consists of several common patterns to reduce complexity and dependencies between classes.

2.1 Template Method

We use the template Method as pattern for sections to define the skeleton of the algorithm in an operation.

2.1.1 Participated Classes and Interfaces

- Section
- ControlSection
- Plattform_Mode_Section
- ImageSection

2.2 Observer Pattern

However the results and states are updated using an observer which defines a one-to-many dependency between objects.

2.2.1 Participated Classes and Interfaces

- Observer
- ObserverManager
- ResultObserver
- StateObserver

2.3 Strategy Pattern

Another interesting pattern to be used by layers is the strategy-pattern. In this context we define a family of algorithm, encapsulate each one and make them interchangeable. This approach also lets the algorithm vary independently from the layer that uses it.

2.3.1 Participated Classes and Interfaces

- ActivationFunction
- Sigmoid
- Tanh
- SoftMax
- ReLu

Different Modes as well as Worker are represented by simple is-relationships.

3 Class Description

- HepicsGui: represents the main entry point to execute the software. It contains the main Method.

- WelcomeWindow: This class is used to display informations about the developed software.

- MainWindow: is the main class that runs the classification system. It is also responsable for setting up the GUI.

- Section: creates a label for each part of the GUI. It has the template method addLabel() which will be implemented by the different sections. Suclasses define sections (e.g. buttons, check boxes) and handle their actions.

- ImageSection: allows the user to add and delete input images. This sections consists of a one combo box which lists all chosen images, three buttons and a thumbnail image.

- Plattform_Mode_Section: allows the user to choose operation mode and a plattform to run the classification or aggregation on it. A checkbox allows the user to choose one or more plattform and a combo box is used to select the mode.

- ControlSection: allows the user to start, pause, resume and cancel the classification process. Thus three buttons are used in this context

  - Once the classification starts, the start button becomes a cancel button.

  - While running the user can pause the classification by clicking on the pause button.

  - When paused the pause button becomes a resume button.

- Observer: defines an updating interface for objects that should be notified of changes in a subject.

- ObserverManager: provides an interface for attaching and detaching Observer objects. It notifies concrete observers whenever corresponding data changes.

- ResultObserver, StateObserver: implement the Observer updating interface to keep its state consistent with concrete subjects and maintain reference to those subjects. These observers might be considered respectively as result section and state section:
  - The result section contains results of aggregation. The results are represented in form of text. The results are the probabilities of the top detected objects in percentages.
  - The state section is mainly the progress bar which shows the state of the current running process.
- Class-Image: is an abstract class which represents the modelisation of an image in the programming language. Its attributes are length, width, channel, and id, which means each uploaded or created image is unique. This class is only a representation of an image object, it can be replaced by an existing library.
  - Attributes:
    * length: is the length of the image object
    * width: is the width of the image object
    * channel: number of channels of the image, by default set to 3 (Red, Green, Blue)
    * id: is a unique number for each image, used to identify them and differentiate them from each other.
  - Methods:
    * getLength(): returns the length of the image
    * setLength(length:int): sets the value of the image's length
    * getWidth(): returns the width of the image
    * setWidth(width:int): sets the value of the image's width
    * getChannel(): returns the number of channels the image has
    * setChannel(channel:int): sets the number of channels of the image
    * resize(length : int, width : int): changes the size of the image to the new dimensions

- Class-InputImage: represents an input image. Has an extra attribute isClassified, which tells if the image has already been classified. In this case, it is not necessary to redo the classification. It is also possible to create a thumbnail out of this image.
  - Attributes:
    * isClassified : a boolean which tells if the image has already been classified. True : display results without classifying, false : start a normal classification process.
  - Methods:
    * createThumbnail() : creates a thumbnail of the input image and returns it as an image. A thumbnail is a smaller representation of the image itself.
    * setStatus(b : boolean) : changes the status of the image to classified or non classified.
    * getStatus() : returns the status of the image : classified/true, not classified/false
- Class-ResultImage : represents the image that is displayed which contains the results of the classification. It has an extra attribute called result, which is a string containing the results. These results can also be displayed in other ways if it is wished to extend the functionality of the system (histograms, pie charts)
  - Attributes:
    * result : a string which contains the results of the classification of an image in percentages.
  - Methods:
    * getResult() : returns the string containing the results of the classification.
    * setResult(results: String) : sets the results of a classification
    * createResult() : draws the results of a classification contained in the string on an image and returns it. This method can be later modified in such way that results would be displayed in a better way.

- Class-ImageManager : is responsible for providing the input image to the system and its preprocessing, which means, converting it into a matrix, or an RGB image, depending on what the process needs.
  - Attributes:
    - input : is the image object representation of the input image.
    - inputPath : is a String which contains the path to the input image
  - Methods:
    - getPath() : returns a string containing the path to the input image
    - setPath(path : String) : sets the path of the input image
    - loadInput(path : String) : loads the input image from its path and returns it
    - convertRGB(input : Image, length : int, width : int, channel : int) : returns an RGB image of the input image for the sake of the classification.
    - preProcessImage : returns the image in the form of a matrix for the sake of the classification.
    - getImage() : returns the input image.

- Class-DataSaver : is practically the data holder. It saves the input images loaded during the time the system has been running in a hashmap. Can also write the results in a text file, and aggregate the results of the given inputs. Aggregating results means in this case calculating the average of multiple results for the corresponding input images. The goal behind it is to achieve higher precision in the classification through selecting more than one image of the same object for the classification.

  - Attributes:

    * results : is a hashmap whose key is the id of the images, and value is the results of the classification of the corresponding image.

  - Methods:

    * addResult(input : InputImage, result : ResultImage) : adds the result of the classification of an input image

    * setResult(input : InputImage, result : ResultImage) : sets the result of the classification of an input image

    * getResult(input : InputImage) : returns the result of the classification of an input image

    * aggregate() : aggregates the results of the last classification process, if there are multiple inputs

    * writeFile(id : int) : writes the results of the corresponding input image to a text file

- Class-ClassificationAssistant : is practically the classifier's secretary. It makes sure that the classifier gets all what it needs to run the classification. It loads weights, sets the input image in the wished form, and loads all the neural networks needed specifications, as well as the classification's class names. Input images get transfered one by one through the assistant.
  - Attributes:
    * inputImage : is the input image
    * weightFilePath : is a String containing the path to the weights file.
    * weights : is an array of double containing the weights of the neural network
    * classNamesPath : is a string containing the path to the class names file. These are the objects to be identified during the classification.
    * isClassified() : is a boolean which tells if the currently loaded input image is classified. By default, it is set on false.
    * manager : is an instance of the class ImageManager. It is responsible for the necessary preprocessing of the input image.
  - Methods:
    * getNeuralNetwork() : returns the currently established neural network
    * setNeuralNetwork(nn : NeuralNetwork) : sets the currently established neural network
    * getInputImage() : returns the currently loaded input image
    * setWeights(weights : Array of double) : sets the array of weights through a new one
    * getWeights() : returns an array of double which contains the weights of the neural network
    * loadClassNames() : returns an array of string containing the class names.
    * getClassNamesPath() : returns a string containing the path to the class names file
    * setClassNamesPath(path : String) : sets the path to the class names file

- Class-NeuralNetwork : is the modelisation of a neural network. A neural network has a name, a topology and a set of layers.
  - Attributes:
    * name : a string containing the name of the neural network
    * topology : an instance of the class topology. It is a characteristic of the neural network
    * layers : is an array of layers that build the neural network
  - Methods:
    * getName() : returns the name of the neural network
    * setName(name : String) : sets the name of the neural network
    * getTopology() : returns the topology object of the neural network
    * setTopology(topology : Topology) : sets the topology of the neural network through a new one
    * addLayer(type : LayerType) : adds a layer to the list of layers of the neural network
    * forwardPropagation (inputImage : Image) : returns a matrix of floats that represent the build through the classification.

- Class-Shape : contains constants which define the dimensions, which means the form and general composition of a layer represented as a matrix.
  - Attributes:
    - rows : the number of rows of the layer and the first dimension of the matrix
    - cols : the number of columns of the layer and the second dimension of the matrix
    - channels : the number of channels of the layer and the third dimension of the matrix
  - Methods:
    - getRow() : returns the number of rows of the layer
    - getCols() : returns the number of columns of the layer
    - getChannels() : returns the number of channels of the layer
- Class-Topology : is the visual representation of the neural network, meaning the layers and the relations or interactions between them.
  - Attributes:
    - displayer : is an instance of the class TopologyDisplayer.
  - Methods:
    - display() : displays the topology of the neural network.

- Class-TopologyDisplayer : is a tool which is responsible for drawing the representation of the topology and generating an image.
  - Attributes:
    * layers : is a list of the layers that constitute the neural network
    * display : is an image on which the topology is drawn
  - Methods:
    * drawTopology() : draws the topology on an image and returns it
    * getDisplay() : returns the image to display
- Class-Layer : is what a neural network is made of. It is a set of nodes placed on each other which each treat the input image depending on their type and activation function.
  - Attributes:
    * type : is an enumeration called LayerType. It defines the type of the layer, and on that note, the way it handles the treatment of the data it receives.
  - Methods:
    * getType() : returns the type of the layer.
    * setType(type : LayerType) : sets the type of the layer
    * getInputShape() : returns the shape of the input
    * setInputShape() : sets the shape of the input
    * getOutputShape() : returns the output's shape
    * setOutputShape() : sets the shape ouf the ouput
    * forwardPropagation(inputImage : Image) consists of processing the data of the input image through one layer, it returns a matrix of float values

- Enumeration Class-LayerType : is an enumeration class of the possible layer types. Each type defines how the layer treats the data it receives and how it gives it over to the next one.
  - InputLayer
  - ConvolutionalLayer
  - MaxPoolLayer
  - ResponseNormalizationLayer
  - DenseLayer
  - FullyConnectedLayer
- InputLayer Class: This class inherits the abstract Layer class. It is the first layer of a neural network model, which contains the input shape of an input image as its attribute.
- Abstract ConvolutionalLayer Class: This class modifies the convolutional layers, which applies a specified number of convolution filters to the image. For each subregion, the layer performs a set of mathematical operations to produce a single value in the output feature map. The filters attribute specifies the number of filters to apply, kernel_size specifies the dimensions of the filters and strides specifies the strides of the convolution along the height and width.
- CPUConvolutionalLayer Class: This class inherits the abstract ConvolutionalLayer Class. Its computation would be executed on the CPU.
- FPGAConvolutionalLayer Class: This class inherits the abstract Convolutional-Layer Class. Its computation would be executed on the FPGA.

- MaxPoolLayer Class: This class modifies the maxpool layer, which downsample the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. It uses max pooling as its pooling algorithms, which extracts subregions of the feature map (e.g., 2x2-pixel tiles), keeps their maximum value, and discards all other values. The pool_size attribute specifies the size of the max pooling filter as [height, width]. If both dimensions have the same value, you can instead specify a single integer and the strides specifies the size of the stride.

- ResponseNormalizationLayer Class: This class modifies the response normalization layer, which normalizes the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. The axis attribute specifies the axis that should be normalized (typically the features axis).

- DenseLayer Class: This class modifies the dense layer, which performs classification on the features extracted by the convolutional layers and downsampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer. The units attribute specifies the number of neurons in the dense layer.

- TrainableLayer Class: This class modifies the layer whose trainable parameters (e.g. weights) can be optimized and updated by the training algorithms (like back-propagation). The weight attribute specifies the weights of this layer and use_bias specifies whether the layer uses a bias vector. The getWeight method returns the weights of the layer as a matrix and setWeight method sets the weights of the layer from a matrix (with the same shapes as the output of getWeight). All the layers implement the forward propagation method in order to use the pre-trained parameters to get the output.

- ActivationFunction Class: This abstract class encapsulates the activation function whose purpose is to allow small changes in the weights or bias to only cause a small change on the output, this property is helpful when training a network. The name attribute specifies the name of the activation function. The activate method specifies the function that would be applied.

- Sigmoid Class: This class encapsulates an activation function called Sigmoid. Its method activate implements the function: $g(z) = 1/(1 + e^{-z})$. It's used on the output layer so that we can easily interpret the output as probabilities since it has restricted output between 0 and 1.

- Tanh Class: This class encapsulates an activation function called Tanh. Its method activate implements the function: $g(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$. It's superior to sigmoid function in which the mean of its output is very close to zero, which in other words center the output of the activation units around zero and make the range of values very small which means faster.

- ReLU Class: This class encapsulates an activation function called ReLU. Its method activate implements the function: $g(z) = max0, z$. The models that are close to linear are easy to optimize. Since ReLU shares a lot of the properties of linear functions, it tends to work well on most of the problems.

- Softmax Class: This class encapsulates an activation function called Softmax. Its method activate generates a value between 0–1 for each node (the sum of all these softmax values is equal to 1). We can interpret the softmax values for a given image as relative measurements of how likely it is that the image falls into each target class.

- ReLU Class: g(z) = max0, z. The models that are close to linear are easy to optimize. Since ReLU shares a lot of the properties of linear functions, it tends to work well on most of the problems.

- FileLoader: A dialog used to open files.

- LayerType: Defines a constant for every type of layer.

- Mode: An interface for classification modes. It has methods for different workers.
  - Methods:
    * nextFileLoadWorkUnit(): returns the next work unit for a file loading worker
    * nextCpuWorkUnit(): returns the next work unit for a cpu worker
    * nextFpgaWorkUnit(): returns the next work unit for a fpga worker
- HighPerformanceMode: A class that implements the methods of interface Mode. Its methods are optimized for high performance.
- LowPowerMode: A class that implements the methods of interface Mode. Its methods are optimized for low power.
- EnergyEfficiencyMode: A class that implements the methods of interface Mode. Its methods are optimized for energy efficiency.
- WorkUnit: An interface for units of work.
  - Methods:
    * run(): runs the unit of work
- ImageLoadWorkUnit: Implements a unit of work for loading images.
- LayerWorkUnit: Implements a unit of work for calculating a Layer of the neuronal network.
- Scheduler: The Scheduler uses the mode to decide which work unit to use next. The mode is set by the respective methods. The Scheduler also holds the RessourceManager. It is able to control the execution and report about its progress.

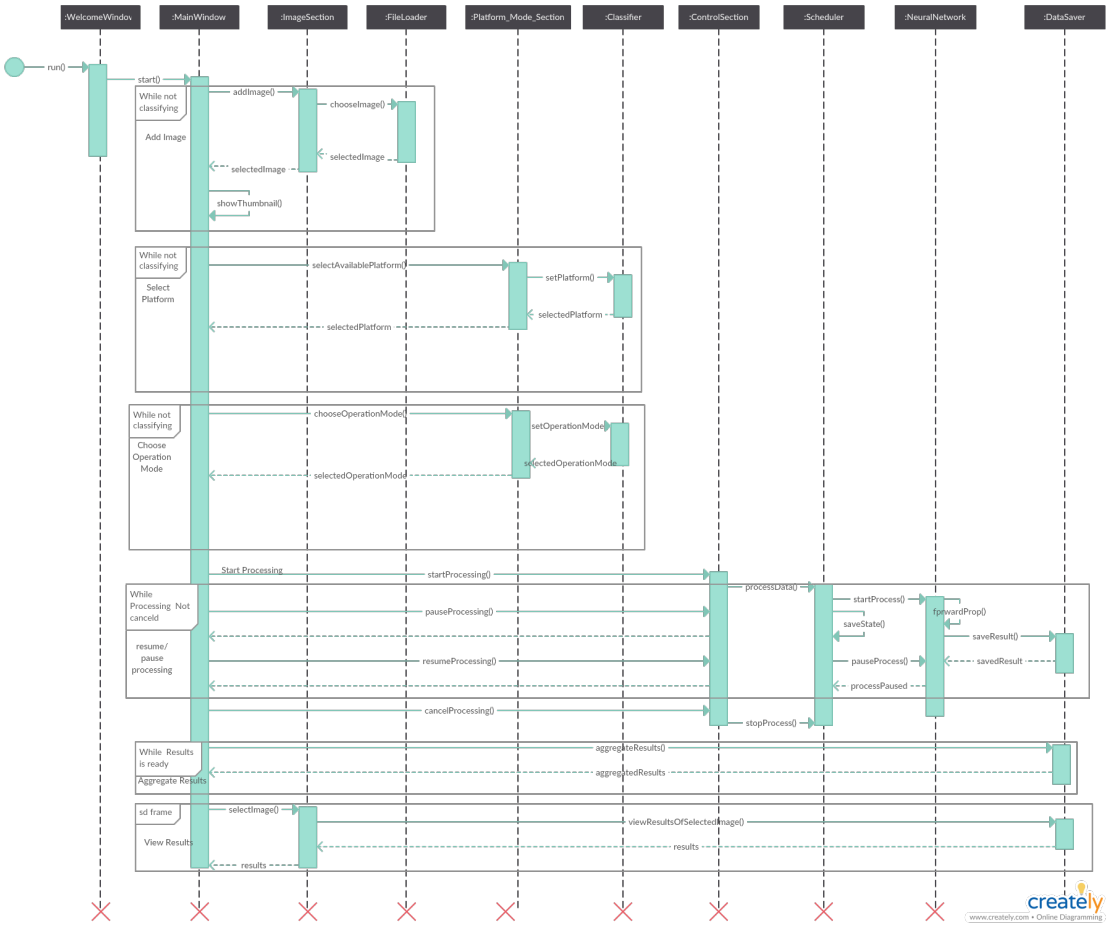
- RessourceManager: Manages ressources for classification. The only ressource at the moment is a counter for files.
  - Attributes:
    - fileTokens: a counter for file tokens
  - Methods:
    - takeFileToken(): decrements the file token counter and blocks if the counter is 0 (like a semaphore)
    - returnFileToken(): increments the file token counter
- Classifer: Controls the classification. Holds the Workers. It creates and deletes workers based on which platforms are enabled.
  - Methods:
    - start(): takes the file list and starts the classification
    - pause(): pauses the classification when running
    - resume(): resumes the classification when paused
    - stop(): stops the classification
    - getProgress(): returns the progress of the classification
    - enableCpu(): when enabling/disabling the cpu platform cpu workers are added/removed
    - enableFpga(): when enabling/disabling the fpga platform fpga workers are added/removed
    - enableGpu(): when enabling/disabling the gpu platform gpu workers are added/removed
    - addWorker(): adds a new worker to the list
    - deleteWorker(): deletes a worker to the list

- Poller: A class for polling a file. The file gets requests for classification. The class extracts the requests.
  - Attributes:
    * inputPath: the path to the request file
    * hasRequest: a boolean that saves if a request has been polled
    * priority: the priority of the next request
  - Methods:
    * poll(): polls the request file
    * getHasRequest(): gets if a request has been polled
    * setHasRequest(): sets if a request has been polled
    * getPath(): gets the path to the request file
    * setPath(): sets the path to the request file
    * getPriority(): gets the priority of the request
    * setPriority(): sets the priority of the request

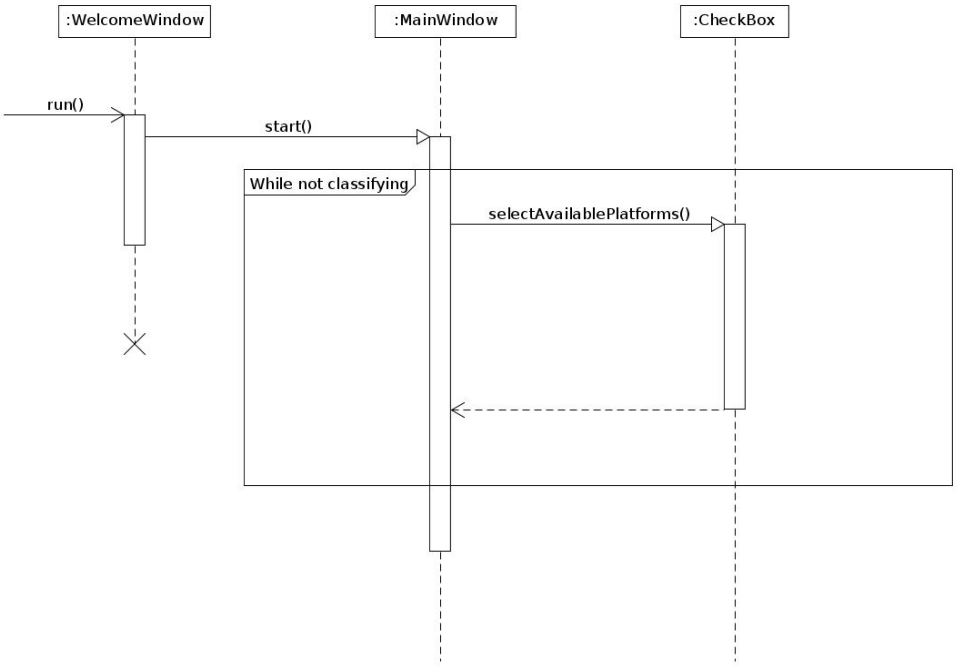
- Worker: The abstraction of a worker. Uses a thread for execution.
  - Attributes:
    - t: the thread used for execution
  - Methods:
    - run(): the main loop of the thread
    - runWorkUnit(): runs the work unit
- GpuWorker: A class implementing the worker for GPU usage.
- FileLoadWorker: A class implementing the worker for loading files.
- CpuWorker: A class implementing the worker for CPU usage.
- FpgaWorker: A class implementing the worker for FPGA usage.

4 Diagrams

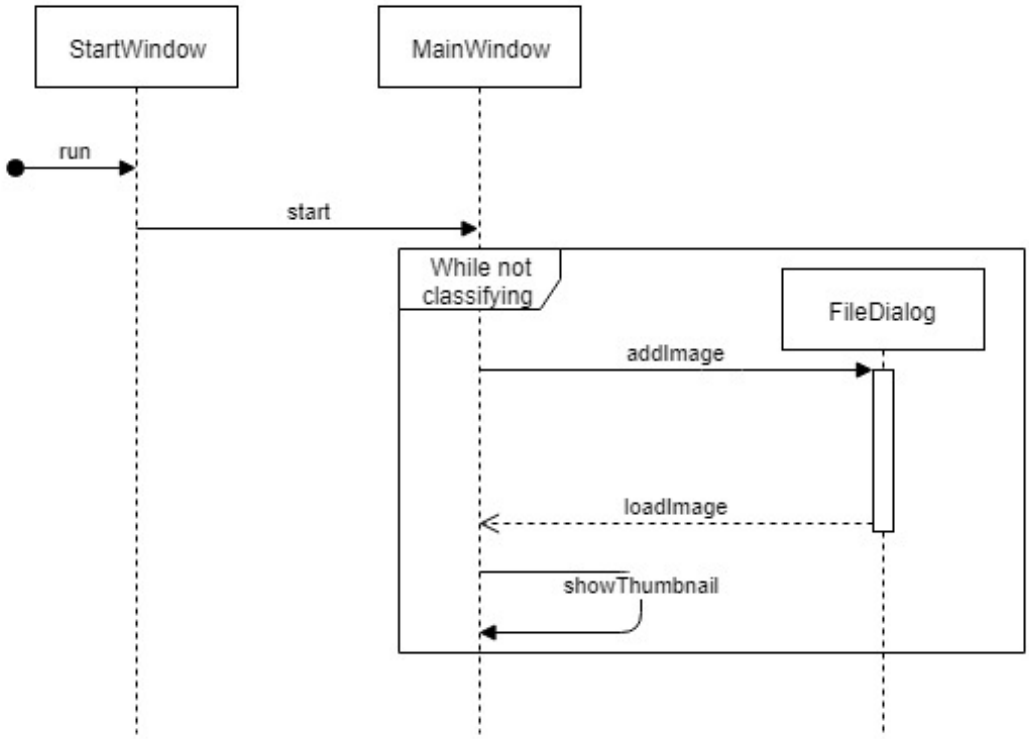
4.1 Overview Sequence Diagram



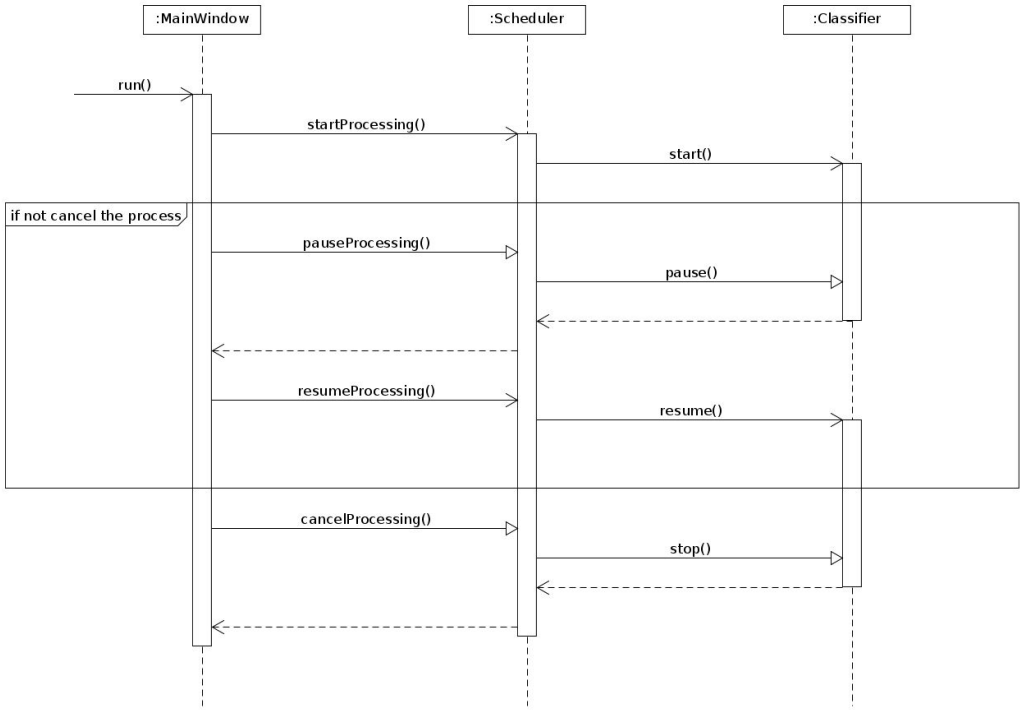
4.2 Select Platforms Sequence Diagram



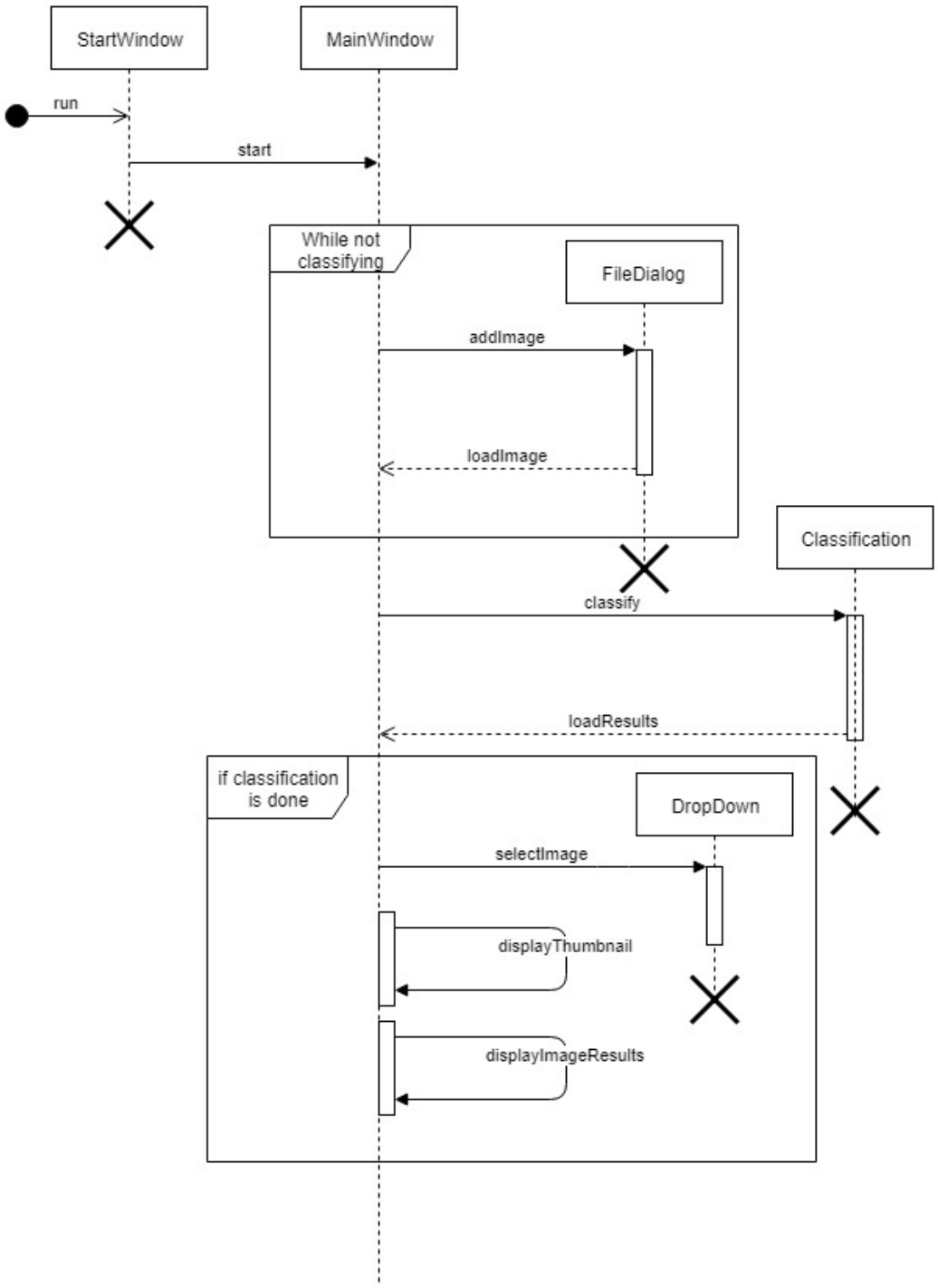
4.3 Select Image Sequence Diagram



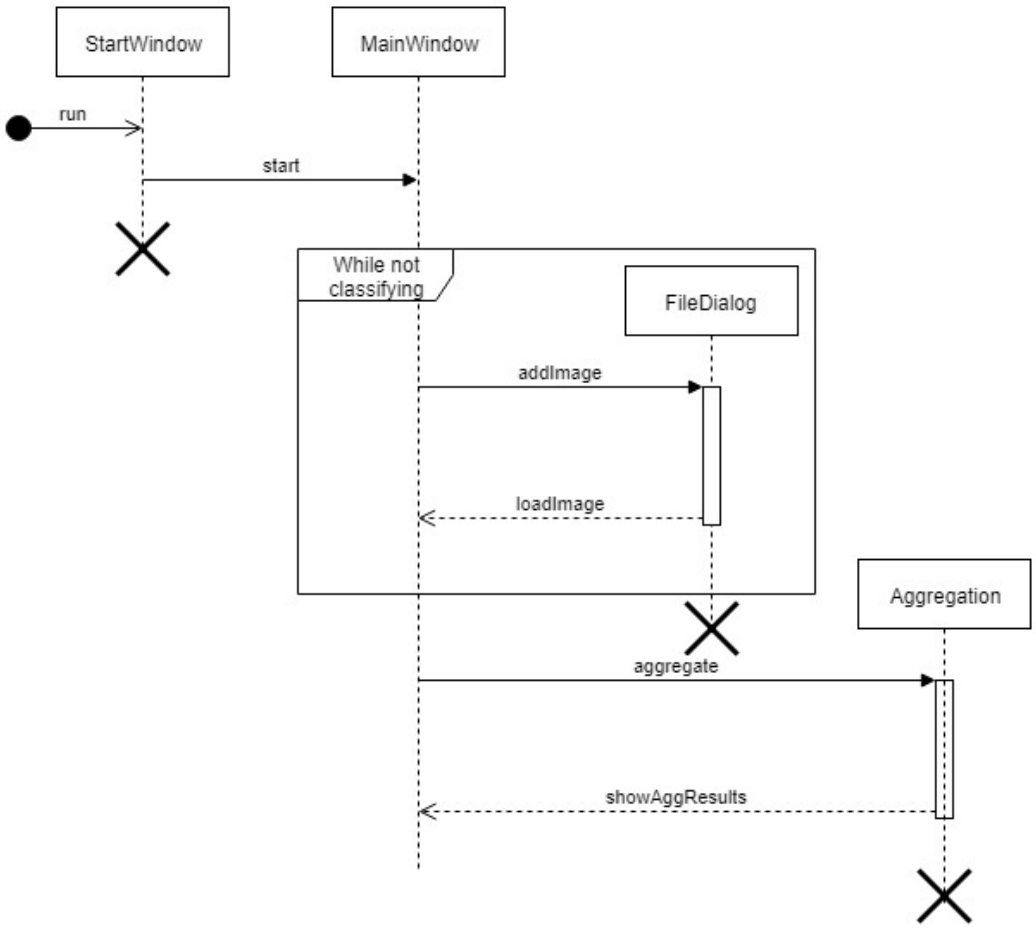
4.4 Classification Sequence Diagram



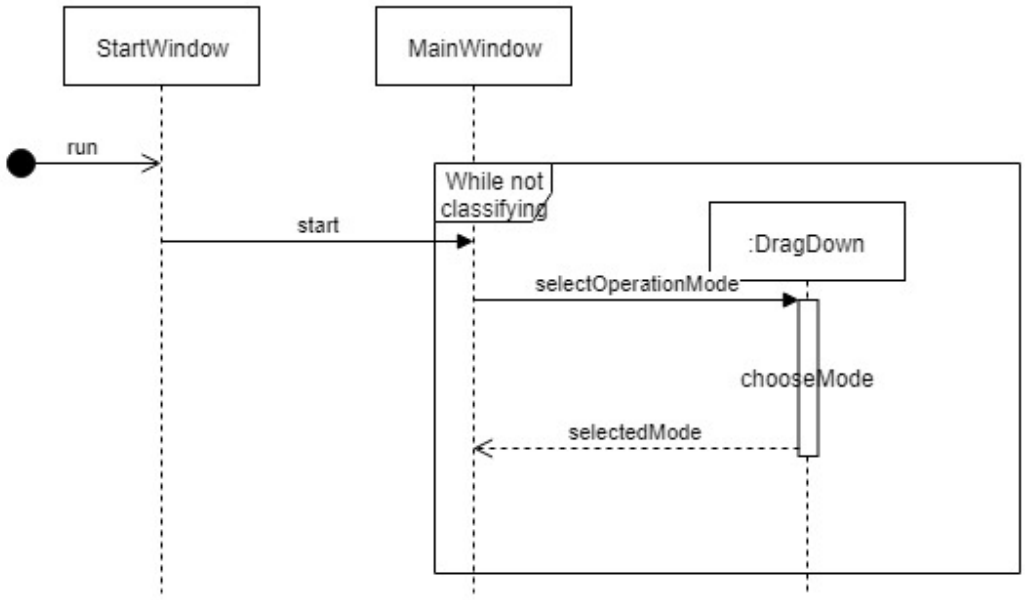
4.5 View Results Sequence Diagram



4.6 Aggregate Sequence Diagram

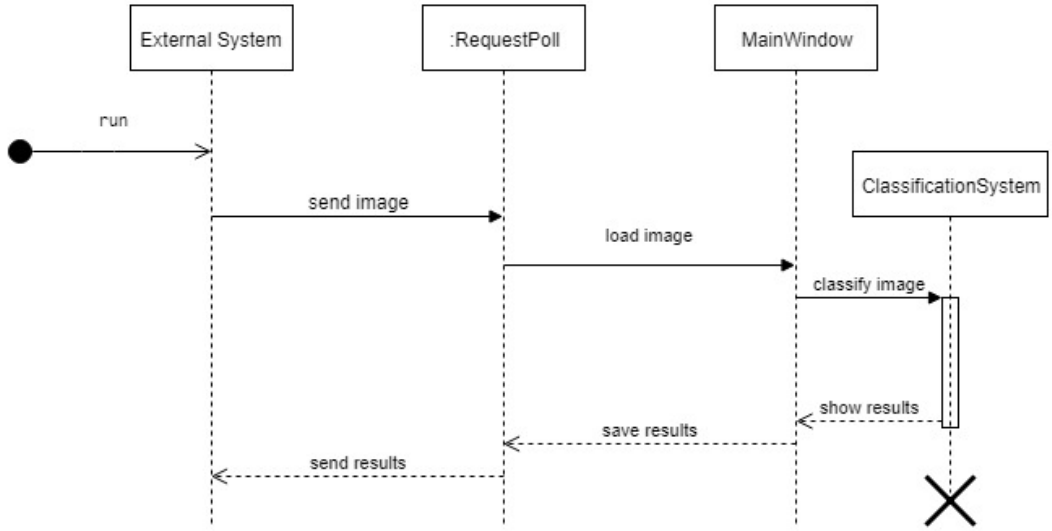


4.7 Select Operation Mode Sequence Diagram



4.8 External Use Sequence Diagram

Diagram.jpg



4.9 State Diagram