HTWK Leipzig
Fachbereich IMN
Vorlesung Softwaretechnik
Sommersemester 05
Datum: 18.04.05

LV Softwaretechnik

Entwurf des Projekts „wxOCR“

Matthias Jauernig
Michael Lahl
03IN1

Gliederung:

1. Einflussfaktoren

Dieser Abschnitt beschreibt die Faktorengruppen, aus denen sich der Aufbau der Software-Architektur des wxOCR-Projektes ableitet.

1.1. Einsatzbedingungen:

- Sequenzieller Programmablauf, d.h. ohne Verteilung oder Nebenläufigkeiten
- Entwurf für den Gebrauch von einem Benutzer zu einem Zeitpunkt an einem Rechner (*single user*), dadurch keine Mehrbenutzerverwaltung nötig

1.2. Umgebungs- / Randbedingungen:

- Umgebungsvorgaben:
  - Hardware: Intel x86 – kompatibel, IBM PC
  - Software:
    - Windows 9x / 2k / XP oder
    - Linux mit X11 und GTK+ installiert
- Integration in bestehende Produkte: keine, es handelt sich um eine komplette „stand-alone-Applikation“, von den Umgebungsvorgaben einmal abgesehen.

1.3. Randbedingungen aus nichtfunktionalen Produkt- / Qualitätsanforderungen:

- Plattformübergreifende GUI für Windows und Linux (ergibt sich aus Pflichtenheft 5.6).
- Anpassbarkeit und Austauschbarkeit vor allem der Bildverarbeitungskomponente.
- Aufteilung in die beiden Teilprogramme „wxOCR (main)“ und „wxOCR netTrainer“, um dem Endbenutzer den langwierigen Prozess des Netz-Anlernens zu ersparen.

2. Grundsatzentscheidungen

Hier sind alle wichtigen Entscheidungen aufgeführt, die beim Entwurf der Software berücksichtigt und getroffen werden sollen. Diese Entscheidungen ergeben sich zum Teil auch aus den Bedingungen von 1.

2.1. Datenhaltung:

- Speicherung in „*flachen*“ Dateien.
- Keine Einbindung in bestehende Datenbanksysteme.

2.2. Verteilung im Netz:

- Keine Verteilung im Netz, keine Netzwerkfunktionalität.
- Lokale Applikation, alle Daten und Prozesse werden lokal behandelt.

2.3. Benutzungsoberfläche:

- Einsatz des UI-Toolkits *wxWidgets*, womit die Bedingung des plattformübergreifenden GUI-Systems erfüllt werden soll.
- Keine Nutzung eines UI-Builder, da solche Tools für wxWidgets zudem nicht gut genug ausgereift sind.
- Die Entscheidung zu wxWidgets lässt sich auch mit der Wahl von C++ als Programmiersprache und von der freien Verfügbarkeit dieser Grafikbibliothek unter der GPL begründen, da wxOCR ebenfalls auf dieser Lizenz beruhen soll.

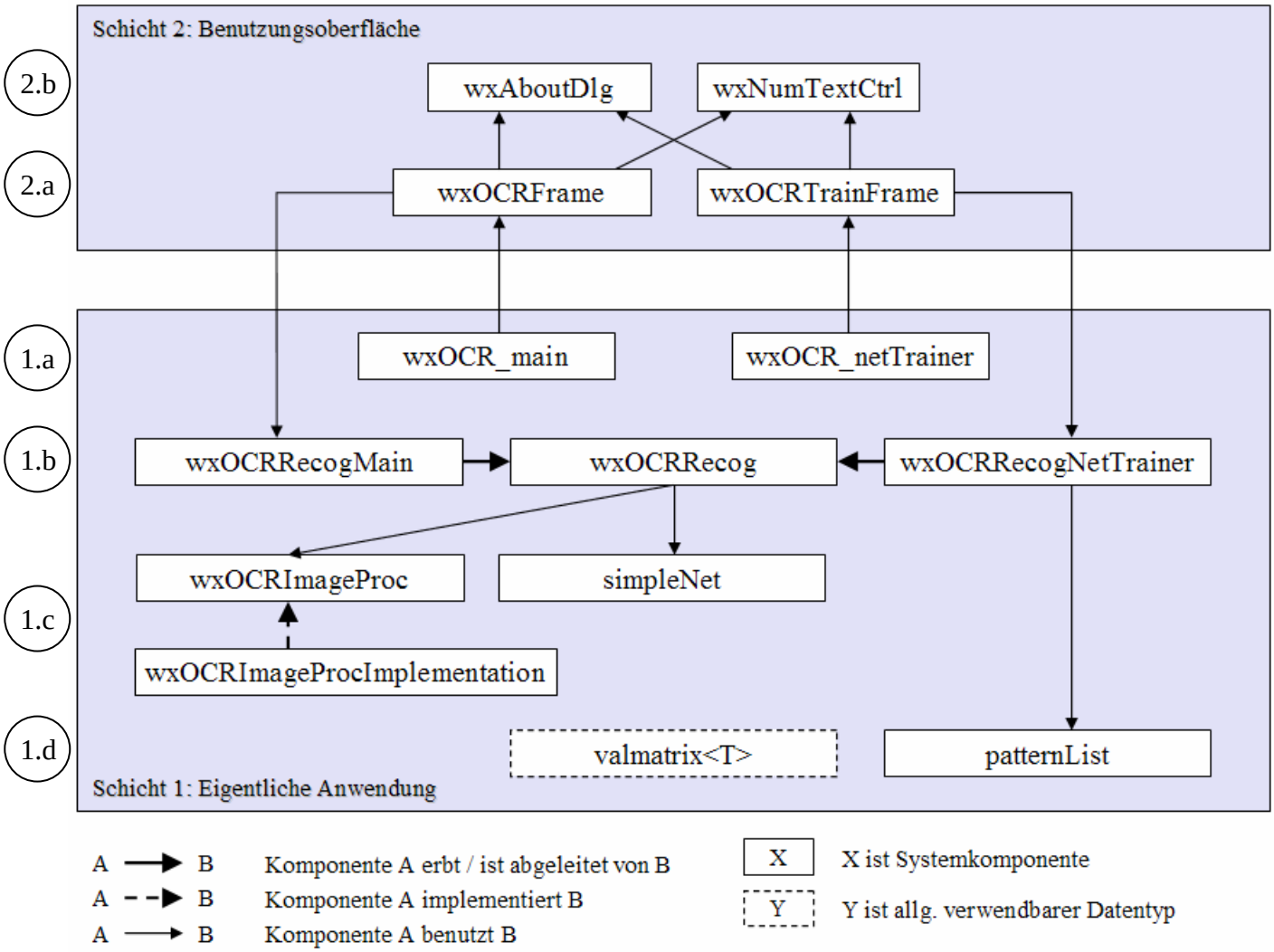
2.4. Hilfesystem:

- Bereitstellung von einfachen HTML-Dateien.
- Realisierung von Entwickler- und Enbenutzer-Hilfen.

3. Software-Architektur

3.1. Schichtenarchitektur:

- 2-Schichten-Architektur, bestehend aus:
  - Schicht 1: eigentliche Anwendung
  - Schicht 2: Benutzungsoberfläche
- Begründung der Schichtenarchitektur-Nutzung:
  - Schichtenarchitektur erlaubt eine übersichtliche Strukturierung.
  - Unterstützt die Forderung nach Änderbarkeit und Anpassbarkeit durch die hohe Modularisierung.
  - Unterstützt die unabhängige Entwicklung von Benutzungsoberfläche und eigentlicher Anwendung durch unterschiedliche Entwickler.
  - Keine 3-Schichten-Nutzung, da die Methoden zum Laden und Speichern von Dateien Teil der Funktionalität und somit in den Funktionalitätsklasse verankert sind, welche zu der eigentlichen Anwendung gehören.
- Grafische Veranschaulichung mit allen Systemkomponenten – bestehend aus den beiden Teilapplikationen wxOCR (main) und wxOCR net Trainer (1.a):



3.2. Feinere Unterteilung der Schichten:

Die beiden Schichten lassen sich noch einmal in kleinere Teilschichten untergliedern, die logisch voneinander getrennt sind.

3.2.1. Schicht 1: Eigentliche Anwendung

1.a) *Module für die Hauptprogramme*: abgeleitet von wxApp, um die Applikation zu erstellen; diese Module dürfen lediglich auf Ebene 2.a) direkt zugreifen.

1.b) *Basisfunktionalität*: Module hier implementieren die Funktionen, die der Benutzer durch Interaktion direkt ausführt, z.B. Netz anlernen, Muster hinzufügen/löschen etc. – die Funktionen dieser Ebene werden direkt und nur von Ebene 2.a) genutzt.

1.c) *Algorithmen*: 2 Hauptbestandteile – Bildverarbeitung und Neuronales Netz; die hier implementierten Algorithmen laufen im Hintergrund ab, sind vom Benutzer also nicht sichtbar – nur die Module der Schicht 1.b) können sie direkt nutzen.

1.d) *Datenstrukturen*: valmatrix ist eine polymorphe Datenstruktur, die eine 2D-Matrix als Klasse erstellt und von den meisten Modulen des Projekts genutzt wird; patternList implementiert eine verkettete Liste, in der die einzelnen Zeichensätze abgespeichert werden.

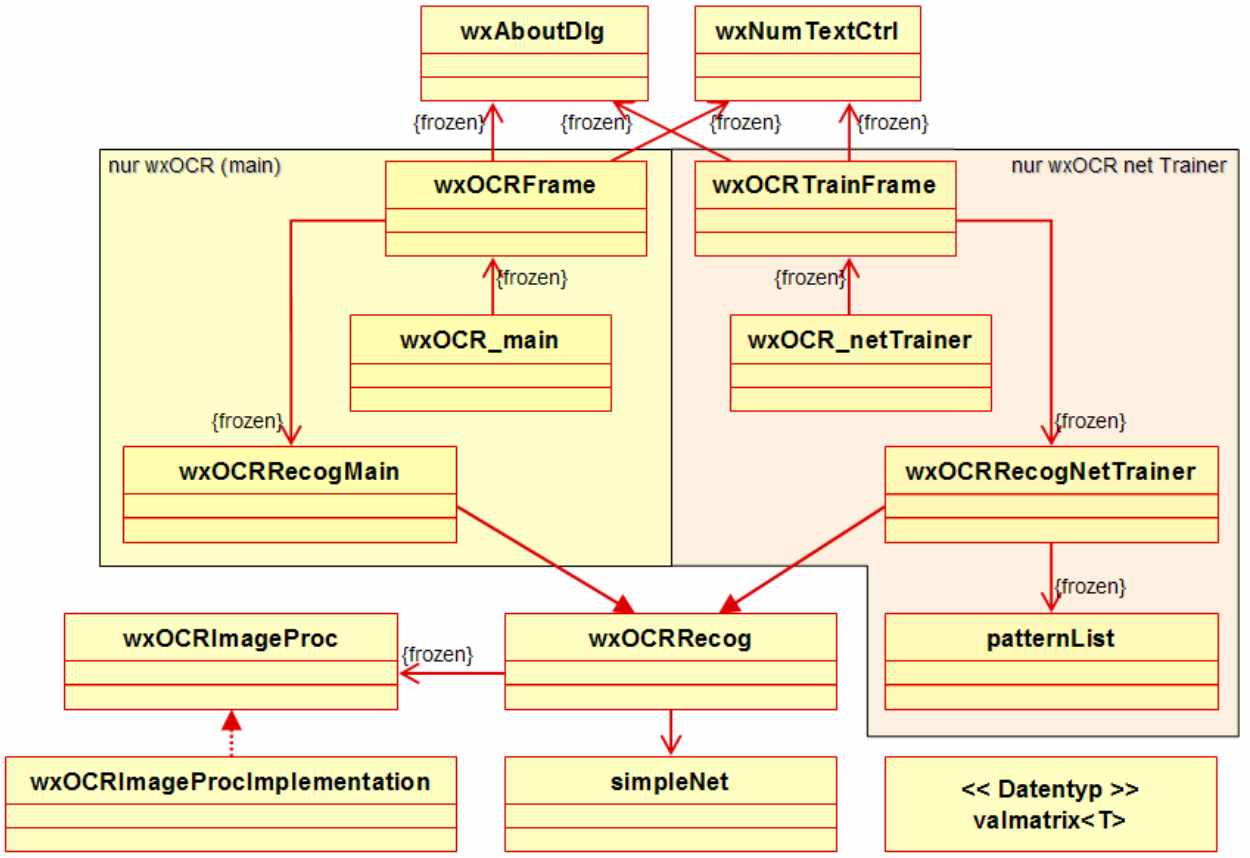
3.2.2. Schicht 2: Benutzungsoberfläche

2.a) *Frames*: hier werden die Hauptfenster mit all ihren Bedienelementen erstellt; diese Schicht kommuniziert nur direkt mit Ebene 1.b) und nutzt deren Funktionalität.

2.b) *Bedienelemente*: diese Module spezifizieren spezielle Bedienelemente bzw. Dialoge, die nicht direkt von dem UI-Toolkit zur Verfügung gestellt werden.

3.3. Klassendiagramm:

Aus der Schichtenarchitektur von 3.1 ergibt sich unmittelbar folgende Darstellung des Klassendiagramms für die Gesamtsoftware:



3.4. Änderungen zum Pflichtenheft:

Das ursprüngliche Klassendiagramm, wie im Pflichtenheft definiert, wurde grundlegend neu gestaltet und modularisiert. So bestehen nun weniger implizite Datenabhängigkeiten, da diese explizit in neue Klassen verlagert wurden. An den ursprünglichen Zielstellungen, Kriterien und Bedingungen hat sich dabei aber nichts verändert.

Gründe für die Modularisierung sind unter anderem:

- Forderung nach Änderbarkeit und Anpassbarkeit: mit der alten Struktur konnte dieser nicht entsprochen werden, durch die größere Modularisierung können nun allerdings leicht einzelne Klassen ausgetauscht und durch andere ersetzt werden. Vor allem die Parametrisierung der wxOCRRecog-Klassen mit einer konkreten Instanz einer Implementierung des abstrakten Bildverarbeitungsmoduls wxOCRImageProc erlaubt es einem Benutzer von Schicht 1, leicht eigene Bildverarbeitungsalgorithmen zu implementieren.
- Weniger Datenredundanzen und -abhängigkeiten: durch die neue Struktur wurden versteckte Datenabhängigkeiten und -redundanzen aufgelöst, so dass nun mehr Übersichtlichkeit und Verständlichkeit gegeben ist.
- Paralleles Laden und Lernen mehrerer Zeichensätze: dies machte es nötig, eine verkettete Liste für die Datentypen zu implementieren, welche in jedem Knoten einen Zeichensatz und einen Identifier für diesen Zeichensatz aufnehmen kann. Da einige Methoden zum Arbeiten mit der verketteten Liste nötig sind, war es nur sinnvoll, diesen Datentypen in eine eigene Klasse auszulagern.

3.5. Beschreibung der Systemkomponenten:

Da es sich hier um ein sehr weiträumiges Gebiet handelt, sollen beim Funktionsumfang lediglich auf die nötigen Daten und auf die öffentlichen Methoden und die Schnittstellen zu anderen Klassen eingegangen werden. Private Methoden sollen hingegen außen vor bleiben und dem jeweiligen Entwickler überlassen werden.

Allgemein zur Implementierung muss noch gesagt werden, dass wir Wert auf ein objektorientiertes Design legen wollen. Da in C++ 2-dimensionale Arrays nicht ad hoc durch eine Klasse dargestellt werden können, haben wir uns entschieden eine eigene Klasse hierfür anzubieten: valmatrix<T>. Diese arbeitet mit valarrays und slices, zwei Datenstrukturen, die von der C++ Standardbibliothek angeboten werden und mit denen es möglich ist, durch geschicktes Arbeiten eine 2D-Matrix als Klasse simulieren zu können. Vorteil der Verwendung von valmatrix ist neben der Nutzung von objektorientierten Konzepten auch das Anbieten von Zusatzfunktionalität. Auf konkrete Matrizenelemente kann man wie gewohnt mittels [][] zugreifen, zusätzlich werden aber auch die Matrixdimensionen gespeichert, die sich explizit mittels Methodenaufrufen bekommen lassen.
Wo wir von diesem Konzept abgewichen sind, ist bei der Implementierung des Neuronalen Netzes durch die Klasse simpleNet. Diese muss so ausgerichtet sein, dass sie beim Anlernen des Netzes so schnell wie möglich arbeitet. Es hat sich allerdings gezeigt, dass sich die Verwendung der 2D-Matrix-Klasse in einer 300%-igen Laufzeit gegenüber der Verwendung von float** niederschlägt. Unsere Vermutung ist, dass ein Compiler eine float**-Matrix effizienter optimieren kann, wohingegen das bei Verwendung einer Klasse nicht der Fall ist. Dies hat uns dazu veranlasst, in der Klasse simpleNet komplett auf valmatrix und valarray zu verzichten.

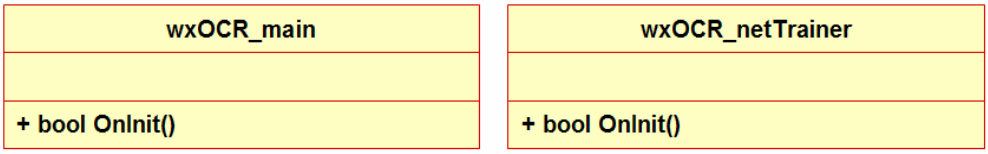
Hier nicht näher angegeben, aber prinzipiell Teil der Programmierarbeit ist die Datenstruktur file_exception aus den Namespaces FILE_ERROR und ANN_ERROR, die vorrangig dann geworfen wird, wenn ein Fehler beim Umgang mit Dateien aufgetreten ist. Wirft eine Methode diese Ausnahme, so wird in der jeweiligen Methoden-Beschreibung darauf hingewiesen.

Wenn nicht anders angegeben, sind bei den folgenden Beschreibungen der Systemkomponenten die Modul-Daten `private`, wohingegen die Modul-Methoden als `public` anzusehen sind.

3.5.1. Schicht 1: Eigentliche Anwendung:

3.5.1.1. wxOCR_main / wxOCR_netTrainer:

- *Detaillierte Darstellung:*

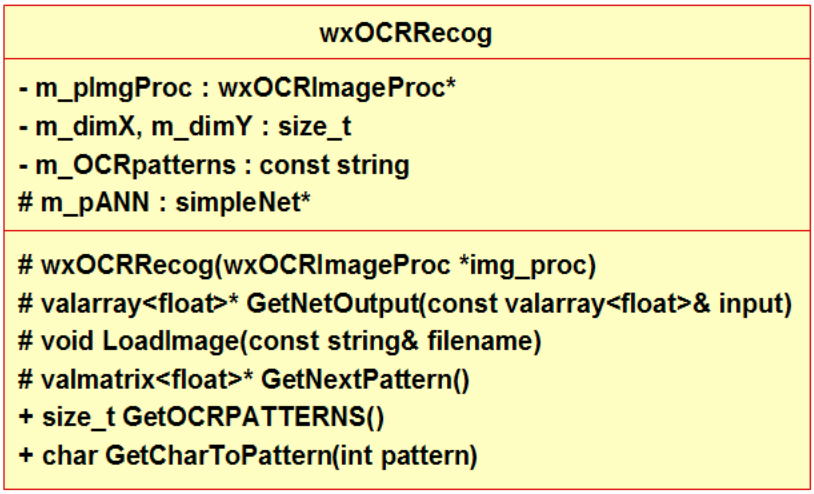


- *Funktionalitätsbeschreibung:*
  Beide Module sind für die jeweilige Anwendungserzeugung verantwortlich, wobei eine Instanz einer Frameklasse erzeugt werden muss, um das entsprechende Fenster angezeigt zu bekommen.

- *Modul-Attribute:*
  - keine
- *Modul-Methoden:*
  - `bool OnInit()`
    Eine von wxApp geerbte Methode, die von beiden Modulen überschrieben wird. Hierin erzeugen sie eine Instanz des konkret darzustellenden Frames, welches bei Programmstart angezeigt wird.

3.5.1.2. wxOCRRecog:

- *Detaillierte Darstellung:*



- *Funktionalitätsbeschreibung:*
  wxOCRRecog bildet die Schnittstelle zwischen wxOCR main, wxOCR net Trainer, der Bildverarbeitungskomponente und dem Neuronalen Netz. Das Modul stellt Instanzen von Klassen und Basisfunktionalitäten zur Verfügung, welche von den beiden Teilapplikationen gemeinsam genutzt werden. Die Funktionalitätsklassen wxOCRRecogMain und wxOCRRecogNetTrainer erben von dieser Klasse und ergänzen sie um ihre applikationsabhängigen Funktionalitäten.

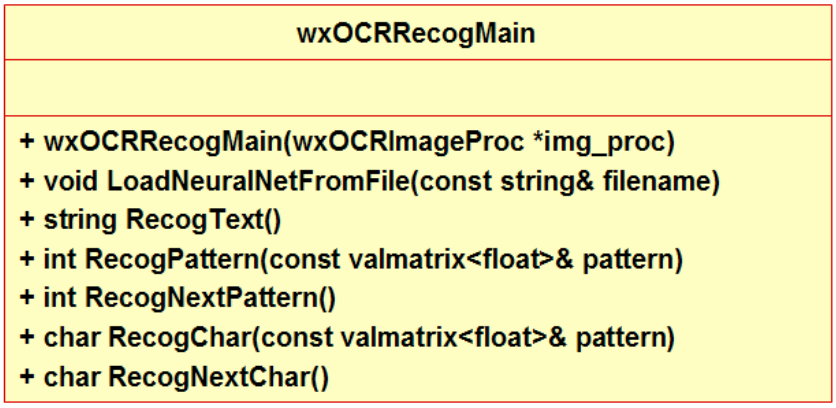
- *Modul-Attribute:*
  - `wxOCRImageProc *m_pImgProc`
    Dies stellt eine konkrete Instanz der Bildverarbeitungskomponente statt. Die Nutzung der abstrakten Klasse wxOCRImageProc verlangt nach der der Initialisierung von m_pImgProc mit einer Instanz einer konkreten Implementierung dieser Klasse. Dies geschieht über den Konstruktor der Klasse wxOCRRecog.
  - `size_t m_dimX, m_dimY`
    Diese Attribute geben die Dimensionen eines einzelnen Musters an, welche benötigt werden, um einen einheitlichen Netz-Input und eine einheitliche Mustergröße nach Extrahierung aus einem Bild zu sichern.
  - `simpleNet *m_pANN`
    Dieses Attribut gibt das Neuronale Netz an, welches für das Anlernen bzw. zum Erkennen von Mustern verantwortlich ist. Es ist protected, sodass abgeleitete Klassen direkt darauf zugreifen können.
  - `const string m_OCRpatterns`
    Diese konstante Zeichenkette stellt alle Zeichen dar, die von dem Programm erkannt und in Text umgewandelt werden sollen.

- *Modul-Methoden:*
  - `wxOCRRecog(wxOCRImageProc *img_proc)`
    Diese Methode ist protected! Der Klassenkonstruktor muss mit einer konkreten Instanz einer Implementierung der Klasse wxOCRImageProc aufgerufen werden. Diese Instanz wird an das entsprechende Attribut zugewiesen, sodass eine Instanz von wxOCRRecog bei der Erstellung mit einem benutzerspezifischen Bildverarbeitungselement parametrisiert wird. Dies erlaubt es einem Benutzer, seine eigenen Bildverarbeitungsklassen zu schreiben und diese ohne Umprogrammieren von wxOCRRecog und darüber liegenden Klassen in das Programm einbinden zu können.
    Vorbedingung: keine
    Nachbedingung: Instanz von wxOCRRecog erzeugt, Initialwerte für m_dimX, m_dimY vergeben und m_pImageProc erzeugt. Hingegen wurde noch keine Instanz von m_pANN angelegt.
  - `valarray<float>* GetNetOutput(const valarray<float>& input)`
    Diese Methode ist protected! Zu dem übergebenen Input wird die Ausgabe des Netzes berechnet und zurückgegeben. Der Aufrufer hat für die Zerstörung des zurückgegebenen valarrays zu sorgen.
    Vorbedingung: Neuronales Netz m_pANN wurde angelegt.
    Nachbedingung: Zum Input berechnete Netzausgabe wurde zurückgegeben.
  - `void LoadImage(const string& filename)`
    Diese Methode ist protected! Ruft die Methode LoadImage() des Attributs m_pImageProc auf, um ein Bild aus dem übergebenen filename zu laden.
    Vorbedingung: keine
    Nachbedingung: Bild wurde geladen. Im Fehlerfall wird eine Exception vom Typ FILE_ERROR::file_exception geworfen.
  - `valmatrix<float>* GetNextPattern()`
    Diese Methode ist protected! Ruft die Methode GetNextPatternWithDims() des Attributs m_pImageProc auf, um das nächste aus dem Bild extrahierte Muster zu erhalten und zurück zu geben.
    Vorbedingung: Bild wurde geladen.
    Nachbedingung: nächstes erkanntes Muster wurde als valmatrix zurückgegeben, NULL bei Fehler, wenn keine weiteren Muster im Bild vorhanden sind oder wenn ein Leerzeichen bzw. Zeilenende erreicht wurde.

- `size_t GetOCRPATTERNS()`
  Liefert die Anzahl der Zeichen zurück, die durch das Neuronale Netz erkannt werden sollen. Leitet sich aus der Länge des Strings m_OCRpatterns ab.
  Vorbedingung: keine
  Nachbedingung: Anzahl an erkennbaren Zeichen zurückgegeben.
- `char GetCharToPattern(int pattern)`
  Liefert das Zeichen aus dem String m_OCRpatterns zurück, welches zum Muster pattern gehört (also das Zeichen m_OCRpatterns[pattern]).
  Vorbedingung: keine
  Nachbedingung: m_OCRpatterns[pattern] zurückgegeben.

3.5.1.3. wxOCRRecogMain:

- *Detaillierte Darstellung:*



- *Funktionalitätsbeschreibung:*
  Diese Klasse leitet sich von wxOCRRecog ab und fügt dieser Klasse Funktionalitäten hinzu, die von wxOCR_main und dessen Benutzungsoberfläche wxOCRFrame benötigt und genutzt werden, wie z.B. das Erkennen von Text aus einem zuvor geladenen Bild.

- *Modul-Attribute:*
  - Keine

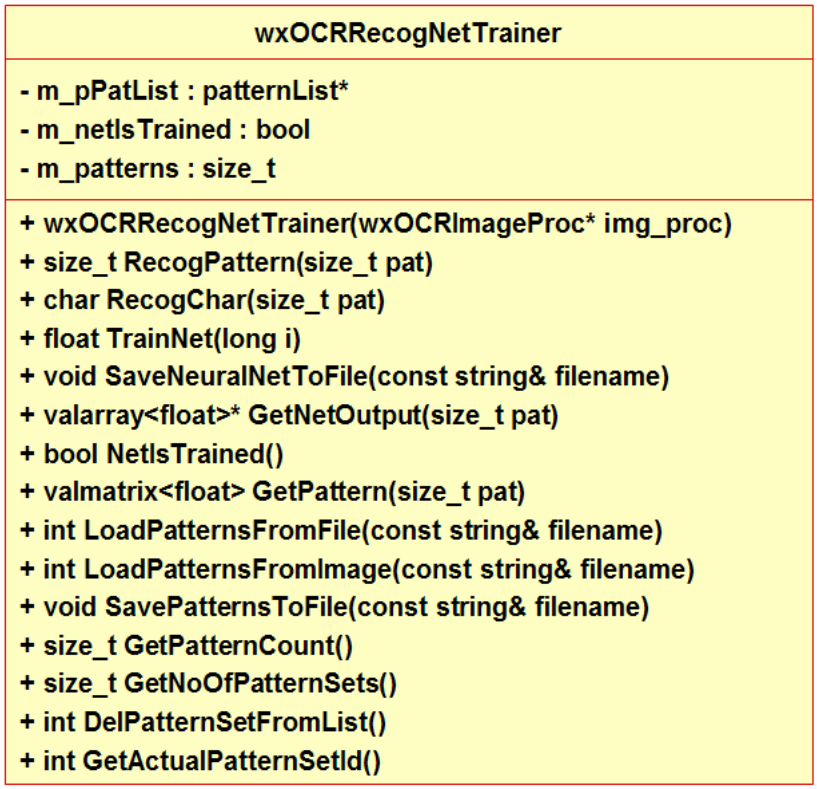
- *Modul-Methoden:*
  - `wxOCRRecogMain(wxOCRImageProc* img_proc)`
    Der Konstruktor der Klasse ruft den Basisklassenkonstruktor wxOCRRecog() auf und parametrisiert eine Instanz der Klasse mit einem konkret erzeugten Bildverarbeitungselement. Dieses kann eine Instanz einer vom Benutzer definierten und von wxOCRImageProc abgeleiteten Klasse sein.
    Vorbedingung: keine
    Nachbedingung: wie wxOCRRecog() unter 3.5.1.2
  - `void LoadNeuralNetFromFile(const string& filename)`
    Diese Methode lädt ein Neuronales Netz durch die mit filename angegebene Datei. Dazu wird der Konstruktor simpleNet(const string& filename) aufgerufen, der eine neue Instanz des Neuronalen Netzes mit den geladenen Netzdaten erzeugt.
    Vorbedingung: keine
    Nachbedingung: m_pANN enthält eine Instanz des aus der Datei filename geladenen Neuronalen Netzes. Im Fehlerfall wird eine FILE_ERROR::file_exception geworfen.
  - `string RecogText()`
    Hierdurch wird der in einem zuvor geladenen Bild enthaltene Text erkannt und als string zurückgegeben.
    Vorbedingung: Neuronales Netz und Bild wurden geladen.

Nachbedingung: Im Bild enthaltener Text wurde durch das Neuronale Netz erkannt und als string zurückgegeben, NULL bei Fehler.

- `int RecogPattern(const valmatrix<float>& pattern)`
  Durch diese Methode wird die Nummer des durch die valmatrix pattern gegebenen Musters zurückgegeben, indem der maximale Output des Neuronalen Netzes beim Input von pattern berechnet wird.
  Vorbedingung: Neuronales Netz wurde geladen.
  Nachbedingung: Nummer des am ehesten erkannten Musters wurde zurückgegeben.
- `int RecogNextPattern()`
  Hierdurch wird das nächste Muster aus dem Bild extrahiert und dann RecogPattern(pattern) aufgerufen, um die Nummer des am ehesten erkannten Musters zurück zu geben.
  Vorbedingung: Neuronales Netz und Bild wurden geladen.
  Nachbedingung: Nächstes Muster wurde aus dem geladenen Bild geholt und die vom Neuronalen Netz am ehesten erkannte Musternummer zurückgegeben.
- `char RecogChar(const valmatrix<float>& pattern)`
  Es wird mit RecogPattern(pattern) das Muster erkannt, welches am ehesten durch pattern angegeben wird. Nachfolgend wird das Zeichen zurückgegeben, welches in m_OCRpatterns mit der Nummer des erkannten Musters korrespondiert.
  Vorbedingung: Neuronales Netz wurde geladen.
  Nachbedingung: Zeichen, welches vom Neuronalen Netz am ehesten erkannt wurde, wird zurückgegeben.
- `char RecogNextChar()`
  Diese Methode extrahiert das nächste Muster aus dem geladenen Bild und gibt das Zeichen zurück, welches vom Neuronalen Netz am ehesten erkannt wurde.
  Vorbedingung: Neuronales Netz und Bild wurden geladen.
  Nachbedingung: Nächstes Muster wurde aus dem Bild geholt und das vom Neuronalen Netz am ehesten erkannte Zeichen zurückgegeben.

3.5.1.4. wxOCRRecogNetTrainer:

- *Detaillierte Darstellung:*



- *Funktionalitätsbeschreibung:*
  Diese Komponente leitet sich von wxOCRRecog ab und fügt dieser Klasse Funktionalitäten hinzu, die von wxOCR_netTrainer und dessen Benutzungsoberfläche wxOCRTrainFrame benötigt und genutzt werden, wie z.B. das Anlernen des Neuronalen Netzes, das Laden und Speichern von Mustern aus Bildern und in Dateien oder das Speichern von Zeichensätzen in einer verketteten Liste.

- *Modul-Attribute:*
  - `patternList *m_pPatList`
    Dies ist die verkettete Liste, die alle zu ladenden Zeichensätze in je einem Knoten aufnimmt. Jeder Zeichensatz wird dabei durch eine eindeutige ID gekennzeichnet.
  - `bool m_netIsTrained`
    Dieses Attribut gibt an, ob das Neuronale Netz bereits trainiert wurde.
  - `size_t m_patterns`
    Dies ist die Anzahl der Muster, die aus einem Bild oder einer Datei als Zeichensatz geladen wurden.

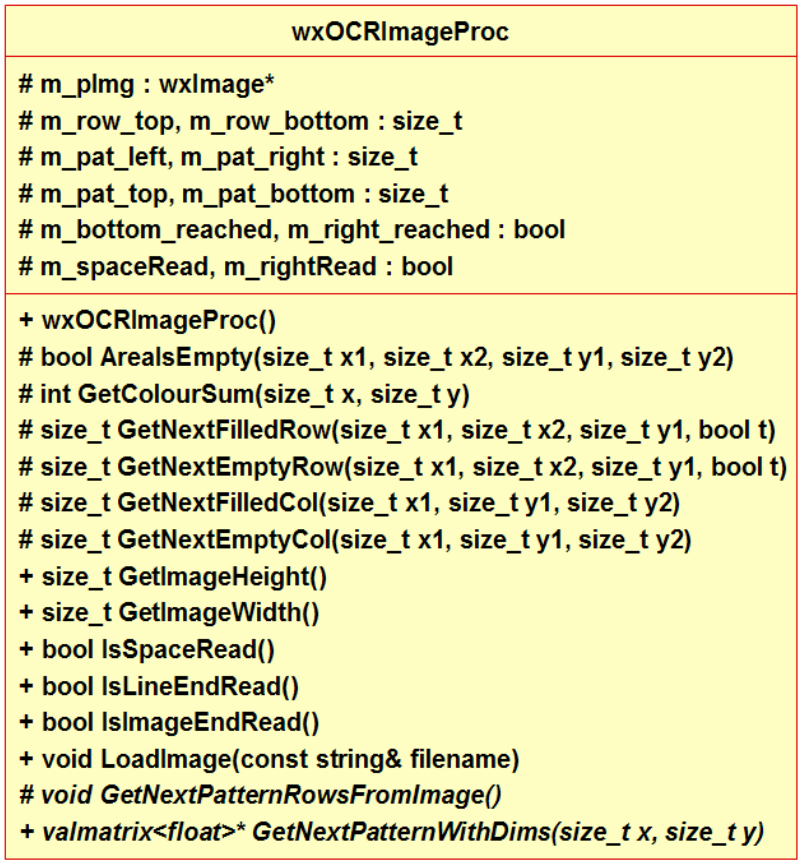
- *Modul-Methoden:*
  - `wxOCRRecogNetTrainer(wxOCRImageProc* img_proc)`
    Der Konstruktor der Klasse ruft den Basisklassenkonstruktor wxOCRRecog() auf und parametrisiert eine Instanz der Klasse mit einem konkret erzeugten Bildverarbeitungselement. Dieses kann eine Instanz einer vom Benutzer definierten und von wxOCRImageProc abgeleiteten Klasse sein.
    Vorbedingung: keine
    Nachbedingung: wie wxOCRRecog() unter 3.5.1.2
  - `size_t RecogPattern(size_t pat)`
    Aus dem aktuellen Zeichensatz wird das Muster pat entnommen und vom Neuronalen Netz der entsprechende Output berechnet. Die Nummer des Neurons mit der höchsten Erkennungsrate wird zurückgegeben.
    Vorbedingung: Neuronales Netz m_pANN wurde erzeugt.
    Nachbedingung: Nummer des am ehesten erkannten Musters wurde zurückgegeben.
  - `char RecogChar(size_t pat)`
    Aus dem aktuellen Zeichensatz wird das Muster pat entnommen und RecogPattern(pat) aufgerufen. Dann wird das zu dem am ehesten erkannten Muster gehörende Zeichen aus m_OCRpatterns entnommen und zurückgegeben.
    Vorbedingung: Neuronales Netz m_pANN wurde erzeugt.
    Nachbedingung: Das am ehesten erkannte Zeichen wurde zurückgegeben.
  - `float TrainNet(long i)`
    Mit dieser Methode wird das Neuronale Netz mit einer Anzahl von i Epochen trainiert, indem die Muster der geladenen Zeichensätze aus der verketteten Liste entnommen und einzeln durch das Neuronale Netz angelernt werden.
    Vorbedingung: Neuronales Netz wurde erzeugt und ein Zeichensatz zum Trainieren ist vorhanden.
    Nachbedingung: Das Neuronale Netz wurde auf alle vorhandenen Zeichensätze i Epochen lang angelernt und der dabei gemachte durchschnittliche quadratische Fehler wurde zurückgegeben.
  - `void SaveNeuralNetToFile(const string& filename)`
    Das vorhandene Neuronale Netz wird in der Datei filename gespeichert, indem von m_pANN die Methode SaveWeightsToFile() aufgerufen wird.
    Vorbedingung: Neuronales Netz zum Speichern vorhanden.
    Nachbedingung: Netz wurde in Datei gespeichert, bei einem Fehler wird eine FILE_ERROR::file_exception geworfen.

- `valarray<float>* GetNetOutput(size_t pat)`
  Das Muster pat aus dem aktuellen Zeichensatz wird als Eingabe des Neuronalen Netzes genommen und damit der Ausgabevektor berechnet, der schließlich auch zurückgegeben wird.
  Vorbedingung: Neuronales Netz m_pANN wurde erzeugt.
  Nachbedingung: Der vom Netz berechnete Output wird als valarray zurückgegeben. Der Aufrufer hat dafür zu sorgen, dass der für das valarray reservierte Speicherplatz wieder freigegeben wird.
- `bool NetIsTrained()`
  Gibt zurück, ob das Netz bereits mit TrainNet() trainiert wurde oder nicht.
  Vorbedingung: keine
  Nachbedingung: Rückgabe des Wertes von m_netIsTrained.
- `valmatrix<float> GetPattern(size_t pat)`
  Das Muster pat des aktuellen Zeichensatzes wird zurückgegeben.
  Vorbedingung: Aktueller Zeichensatz in verketteter Liste vorhanden.
  Nachbedingung: Das gewünschte Muster wurde zurückgegeben.
- `int LoadPatternsFromFile(const string& filename)`
  Die Methode lädt einen Zeichensatz aus der übergebenen Datei filename und legt in der verketteten Liste einen neuen Knoten für diesen Zeichensatz an.
  Vorbedingung: keine
  Nachbedingung: Zeichensatz wurde in verkettete Liste geladen, aktueller Knoten wurde auf diesen Zeichensatz gesetzt und die ID des neu geladenen Zeichensatzes wurde zurückgegeben. Eine FILE_ERROR::file_exception wird geworfen im Fehlerfall.
- `int LoadPatternsFromImage(const string& filename)`
  Die Methode lädt einen Zeichensatz aus dem übergebenen Bild filename und legt in der verketteten Liste einen neuen Knoten für diesen Zeichensatz an.
  Vorbedingung: keine
  Nachbedingung: Zeichensatz wurde in verkettete Liste geladen, aktueller Knoten wurde auf diesen Zeichensatz gesetzt und die ID des neu geladenen Zeichensatzes wurde zurückgegeben. Eine FILE_ERROR::file_exception wird geworfen im Fehlerfall.
- `void SavePatternsToFile(const string& filename)`
  Der aktuelle Zeichensatz wird in der Datei filename gespeichert.
  Vorbedingung: Aktueller Zeichensatz muss vorhanden sein.
  Nachbedingung: Zeichensatz wurde in Datei gespeichert. Im Fehlerfall wird eine FILE_ERROR::file_exception geworfen.
- `size_t GetPatternCount()`
  Die Anzahl der Muster in den geladenen Zeichensätzen wird zurückgegeben.
  Vorbedingung: Zeichensatz muss geladen sein.
  Nachbedingung: m_patterns wurde zurückgegeben.
- `size_t GetNoOfPatternSets()`
  Die Anzahl der geladenen Zeichensätze (also der Knoten in der verketteten Liste) wird zurückgegeben.
  Vorbedingung: keine
  Nachbedingung: Rufe von m_pPatList die Methode GetNodeCountFromList() auf und liefere das Ergebnis zurück.
- `int DelPatternSetFromList()`
  Die Methode löscht den aktuellen Zeichensatz aus der verketteten Liste.
  Vorbedingung: Es muss ein aktueller Zeichensatz in m_pPatList gesetzt sein.
  Nachbedingung: Der aktuelle Zeichensatz wurde gelöscht, ein nächster aktueller Zeichensatz wurde ausgewählt und die ID dieses nächsten Zeichensatzes wurde zurückgegeben.

- `int GetActualPatternSetId()`
  Diese Methode liefert die ID des aktuellen Zeichensatzes zurück.
  Vorbedingung: Es muss ein aktueller Zeichensatz in m_pPatList gesetzt sein.
  Nachbedingung: Die ID des aktuellen Zeichensatzes wurde zurückgegeben.

3.5.1.5. wxOCRImageProc:

- *Detaillierte Darstellung:*



- *Funktionalitätsbeschreibung:*
  Diese Klasse ist abstrakt und stellt eine Vorlage dar für konkrete Implementierungen der Bildverarbeitungskomponente. Diese müssen die abstrakten Methoden GetNextPatternWithDims() und GetNextPatternRowsFromImage() implementieren. Durch Parametrisierung der wxOCRRecog-Klassen mit konrekten Instanzen dieser Implementierungen kann ein Benutzer dieser Klassen leicht die Bildverarbeitungskomponente gegen eigene, vielleicht bessere Implementierungen von wxOCRImageProc austauschen, wodurch der Aspekt der Änderbarkeit erfüllt wird.
  wxOCRImageProc stellt bereits viele Methoden zur Verfügung, die von den darauf basierenden Implementierungen genutzt werden können, so z.B. das Finden nächster leerer bzw. gefüllter Zeilen oder auch die Rückgabe der Farbsumme eines Pixels im mit LoadImage() ladbaren Bild.

  Konkrete Implementierungen müssen nur 2 Methoden implementieren, da nur diese wichtig sind, um Zeichen aus Bildern erkennen zu können.
  Zum einen ist das die Methode GetNextPatternRowsFromImage(), die die nächsten Koordinaten einer Textzeile liest. Um z.B. schiefe Textzeilen lesen zu können, wird die Bearbeitung dieser Methode dem Implementierer überlassen.
  Genauso sieht es mit GetNextPatternWithDims() aus. Dies ist die einzige öffentliche abstrakte Methode und liefert das nächste Zeichen des Bildes mit der gewünschten Größe zurück. Diese Funktionalität reicht bereits aus, um einheitlich Muster zu extrahieren.

- *Modul-Attribute:*
  - `wxImage *m_pImg`
    Dieses Attribut ist protected! Attribut ist das zu ladende und zu verarbeitende Bild.
  - `size_t m_row_top, m_row_bottom, m_pat_left, m_pat_right, m_pat_top, m_pat_bottom`
    Diese Attribute sind protected! Diese Attribute geben die Zeilenbegrenzungen von einer Textzeile bzw. die Begrenzungen eines Musters im Bild an.
  - `bool m_bottom_reached, m_right_reached`
    Diese Attribute sind protected! Diese Attribute geben an, ob der untere bzw. rechte Rand des Bildes beim Durchlaufen bereits gelesen wurde.
  - `bool m_spaceRead, m_lineEndRead`
    Diese Attribute sind protected! Diese Attribute geben an, ob ein Leerzeichen bzw. ein Zeilenende gelesen wurden.

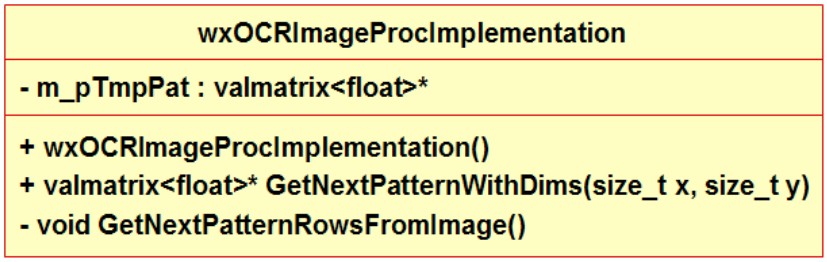
- *Modul-Methoden:*
  - `wxOCRImageProc()`
    Konstruktor, der nur m_pImg auf NULL setzt. Instanzen von wxOCRImageProc können damit nicht erzeugt werden, da die Klasse abstrakte Methoden enthält.
    Vorbedingung: keine
    Nachbedingung: m_pImg=NULL, von Implementierungen von wxOCRImageProc wurde eine Instanz erstellt.
  - `bool AreaIsEmpty(size_t x1, size_t x2, size_t y1, size_t y2)`
    Diese Methode ist protected! Das durch die Punkte (x1,y1) und (x2,y2) definierte Rechteck des geladenen Bildes wird daraufhin untersucht, dass seine Pixel weiß sind bzw. unter einer bestimmten Farbtoleranz liegen.
    Vorbedingung: Bild wurde geladen.
    Nachbedingung: true wird zurückgegeben, wenn der untersuchte Bereich im Bild leer ist, false sonst.
  - `int GetColourSum(size_t x, size_t y)`
    Diese Methode ist protected! Es werden die Farben des Pixels (x,y) im Bild aufsummiert und die so entstandene Summe zurückgegeben. Jede Farbe kann dabei die Werte 0 bis 255 annehmen (8 Bit pro Farbe).
    Vorbedingung: Bild wurde geladen.
    Nachbedingung: Die Farbsumme bei Pixel (x,y) wurde zurückgegeben.
  - `size_t GetNextFilledRow(size_t x1,size_t x2,size_t y1,bool t)`
    Diese Methode ist protected! Sie gibt die Nummer der nächsten von y1 ausgehenden und mit x1 und x2 eingegrenzten gefüllten (Teil-) Zeile des Bildes zurück. „Gefüllt“ heißt dabei, dass mindestens ein Pixel der Zeile über der Farbtoleranz liegt. Ist bool t = true, so wird von oben nach unten durchlaufen, ansonsten von unten nach oben.
    Vorbedingung: Bild wurde geladen.
    Nachbedingung: Nummer der nächsten gefüllten (Teil-) Zeile wurde zurückgegeben.
  - `size_t GetNextEmptyRow(size_t x1,size_t x2,size_t y1,bool t)`
    Diese Methode ist protected! Sie gibt die Nummer der nächsten von y1 ausgehenden und mit x1 und x2 eingegrenzten leeren (Teil-) Zeile des Bildes zurück. „Leer“ heißt dabei, dass alle Pixel einer Zeile unter der Farbtoleranz liegen. Ist bool t = true, so wird von oben nach unten durchlaufen, ansonsten von unten nach oben.
    Vorbedingung: Bild wurde geladen.
    Nachbedingung: Nummer der nächsten leeren (Teil-) Zeile wurde zurückgegeben.
  - `size_t GetNextFilledCol(size_t x1, size_t y1, size_t y2)`
    Diese Methode ist protected! Sie gibt die Nummer der nächsten von x1 ausgehenden und mit y1 und y2 eingegrenzten gefüllten (Teil-) Spalte des Bildes zurück. „Gefüllt“ heißt dabei, dass mindestens ein Pixel der Spalte über der Farbtoleranz liegt.
    Vorbedingung: Bild wurde geladen.
    Nachbedingung: Nummer der nächsten gefüllten (Teil-) Spalte wurde zurückgegeben.

- `size_t GetNextEmptyCol(size_t x1, size_t y1, size_t y2)`
  Diese Methode ist protected! Sie gibt die Nummer der nächsten von x1 ausgehenden und mit y1 und y2 eingegrenzten leeren (Teil-) Spalte des Bildes zurück. „Leer“ heißt dabei, dass alle Pixel einer Spalte unter der Farbtoleranz liegen.
  Vorbedingung: Bild wurde geladen.
  Nachbedingung: Nummer der nächsten leeren (Teil-) Spalte wurde zurückgegeben.
- `virtual void GetNextPatternRowsFromImage() = 0`
  Diese Methode ist protected! Sie ist abstrakt und muss von konkreten Implementierungen von wxOCRImageProc definiert werden. Sie schreibt in die Attribute m_row_top und m_row_bottom die Rahmengrößen der nächsten Textzeile des Bildes.
  Vorbedingung: Bild wurde geladen und es wurde noch nicht das Bildende erreicht.
  Nachbedingung: m_row_top und m_row_bottom wurden mit den Koordinaten der nächsten Textzeile beschrieben.
- `size_t GetImageHeight()`
  Hierdurch wird die Höhe des Bildes in Pixeln zurückgegeben.
  Vorbedingung: Bild wurde geladen.
  Nachbedingung: Bildhöhe wurde zurück geliefert.
- `size_t GetImageWidth()`
  Hierdurch wird die Breite des Bildes in Pixeln zurückgegeben.
  Vorbedingung: Bild wurde geladen.
  Nachbedingung: Bildbreite wurde zurück geliefert.
- `bool IsSpaceRead()`
  Hat ein Aufruf von GetNextPatternWithDims() NULL zurück geliefert, so kann mit dieser Methode getestet werden, ob ein Leerzeichen Grund dafür war.
  Vorbedingung: Bild wurde geladen und mit GetNextPatternWithDims() bereits ausgelesen.
  Nachbedingung: Sollte ein Leerzeichen gelesen worden sein, so wird true zurückgegeben, false sonst. Zudem wird m_spaceRead auf false zurückgesetzt.
- `bool IsLineEndRead()`
  Hat ein Aufruf von GetNextPatternWithDims() NULL zurück geliefert, so kann mit dieser Methode getestet werden, ob ein Zeilenende Grund dafür war.
  Vorbedingung: Bild wurde geladen und mit GetNextPatternWithDims() bereits ausgelesen.
  Nachbedingung: Sollte ein Zeilenende gelesen worden sein, so wird true zurückgegeben, false sonst. Zudem wird m_lineEndRead auf false zurückgesetzt.
- `bool IsImageEndRead()`
  Hat ein Aufruf von GetNextPatternWithDims() NULL zurück geliefert, so kann mit dieser Methode getestet werden, ob das Erreichen des Bildendes Grund dafür war.
  Vorbedingung: Bild wurde geladen und mit GetNextPatternWithDims() bereits ausgelesen.
  Nachbedingung: Sollte das Erreichen des Bildendes Grund für die Rückgabe von NULL gewesen sein, so wird true zurückgegeben, false sonst.
- `void LoadImage(const string& filename)`
  Diese Methode lädt das durch die Datei filename angegebene Bild in das Attribut m_pImg und setzt die Attribute, die Bildkoordinaten darstellen, entsprechend.
  Vorbedingung: keine
  Nachbedingung: Bild wurde geladen, bei einem Fehler wird eine FILE_ERROR::file_exception geworfen.
- `virtual valmatrix<float>* GetNextPatternWithDims(size_t x, size_t y) = 0`
  Diese Methode ist abstrakt. Sie muss von konkreten Implementierungen von wxOCRImageProc definiert werden. Sie ist dazu da, um aus dem geladenen Bild das nächste Muster als valmatrix mit den Dimensionen x und y zurück zu geben. Der

Aufrufer der Methode hat dafür zu sorgen, dass der für das zurückgegebene Muster reservierte Speicherplatz wieder freigegeben wird.
Vorbedingung: Bild wurde geladen.
Nachbedingung: Nächstes Muster aus dem Bild wurde zurückgegeben bzw. NULL, wenn entweder keine weiteren Muster im Bild vorhanden sind oder eine Leerzeile, ein Leerzeichen bzw. ein Zeilenende gelesen wurden.

3.5.1.6. wxOCRImageProcImplementation:

- *Detaillierte Darstellung:*



- *Funktionalitätsbeschreibung:*
  Diese Klasse stellt EINE konkrete Implementierung von wxOCRImageProc dar. Sie ist von wxOCRImageProc abgeleitet und implementiert deren abstrakte Methoden GetNextPatternWithDims() und GetNextPatternRowsFromImage(), um Muster aus Bildern extrahieren zu können.

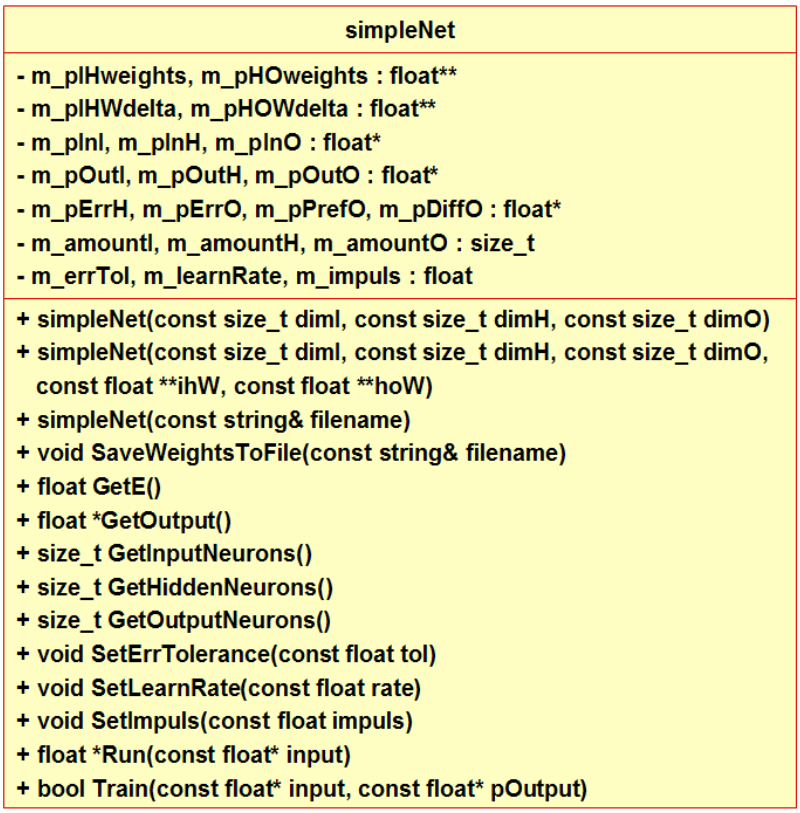
- *Modul-Attribute:*
  - `valmatrix<float> *m_pTmpPat`
    Dieses Attribut stellt eine Matrix mit dem zuletzt gelesenen Muster dar. Dieses wird benötigt, um bei Kerning den linken Rand des nächsten Musters erkennen zu können.

- *Modul-Methoden:*
  - `wxOCRImageProcImplementation()`
    Der Konstruktor ruft den Basisklasse-Konstruktor wxOCRImageProc() auf, um z.B. das zu ladende Bild-Attribut m_pImg auf NULL zu setzen.
    Vorbedingung: keine
    Nachbedingung: m_pImg auf NULL gesetzt, Bildverarbeitungsinstanz erzeugt.
  - `valmatrix<float>* GetNextPatternWithDims(size_t x, size_t y)`
    Dies implementiert die abstrakte Methode der Klasse wxOCRImageProc. Hierdurch wird das nächste Muster aus dem Bild erkannt und als valmatrix mit den Dimensionen x und y zurückgegeben. Der Aufrufer der Methode hat dafür zu sorgen, dass der für das zurückgegebene Muster reservierte Speicherplatz wieder freigegeben wird.
    Vorbedingung: Bild wurde geladen.
    Nachbedingung: Das nächste aus dem Bild extrahierte Muster wurde mit den Dimensionen x und y zurückgegeben. Wenn kein weiteres Bild vorhanden ist, wird NULL zurückgegeben, ebenfalls wenn ein Leerzeichen bzw. ein Zeilenende / eine Leerzeile gelesen wurde.
  - `void GetNextPatternRowsFromImage()`
    Diese Methode ist private! Sie implementiert die abstrakte Methode der Klasse wxOCRImageProc. Hierdurch wird die nächste Textzeile mit enthaltenen Mustern eingerahmt durch Setzen der Attribute m_row_top und m_row_bottom, sodass die Muster in dieser Zeile extrahiert werden können.
    Vorbedingung: Bild geladen und Bildende noch nicht erreicht.

Nachbedingung: m_row_top und m_row_bottom mit den Koordinaten der nächsten Textzeile belegt.

3.5.1.7. simpleNet:

- *Detaillierte Darstellung:*



- *Funktionalitätsbeschreibung:*
  Diese Komponente realisiert ein einfaches vorwärtsgerichtetes Neuronales Netz. Dieses Netz besteht aus 3 Schichten und lernt überwacht mit dem Backpropagation-Algorithmus. Dazu kann ein Benutzer der Klasse die Methode Train() mit einem Input und von ihm gewünschten Output aufrufen und das Netz ändert seine Gewichte dahin gehend, dass der Fehler zwischen gewolltem und tatsächlichem Output minimal wird.
  Nach der Trainingsphase muss man nur noch Run() mit einem Inputvektor aufrufen und das Netz berechnet seine Ausgabe anhand der bis dahin angepassten Gewichtsmatrizen.
  Um die Trainingsphase so schnell wie möglich zu absolvieren, wurde statt einer 2D-Matrix-Klasse eine float**-Matrix genommen, wodurch sich die Performance deutlich steigern lässt.

- *Modul-Attribute:*
  - `float **m_pIHweights, **m_pHOweights`
    Diese Attribute geben die Gewichtsmatrizen zwischen der Eingabe- und der verborgenen Schicht bzw. zwischen verborgener und Output-Schicht an.
  - `float **m_pIHWdelta, **m_HOWdelta`
    Diese Attribute geben die Gewichtsänderungsmatrizen zwischen der Eingabe- und der verborgenen Schicht bzw. zwischen verborgener und Output-Schicht an.
  - `float *m_pInI, *m_pInH, *m_pInO`
    Diese Vektoren geben den jeweiligen Input für die Neuronen der 3 Schichten an.
  - `float *m_pOutI, *m_pOutH, *m_pOutO`
    Diese Vektoren geben den jeweiligen Output der Neuronen der 3 Schichten an.

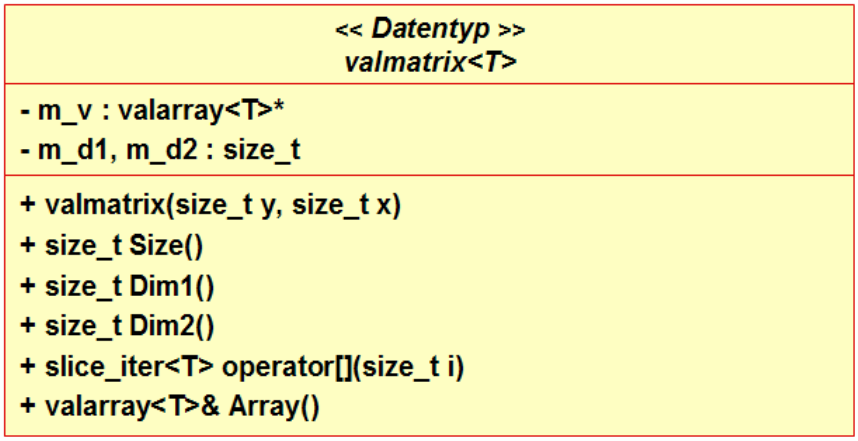
- `float *m_pErrH, *m_pErrO, *m_pPrefO, *m_pDiffO`
  Diese Vektoren geben den gemachten Fehler von verborgener (m_errH) und Output-Schicht (m_errO) an bzw. den gewollten Output des Netzes (m_prefO) sowie die Differenz zwischen gewolltem und tatsächlichem Output (m_diffO).
- `size_t m_amountI, m_amountH, m_amountO`
  Diese Attribute geben die Anzahl der Neuronen pro Schicht an, also die Dimensionen des Neuronalen Netzes.
- `float m_errTol, m_learnRate, m_impuls`
  Durch diese Attribute wird dei Fehlertoleranz, die Lernrate und des gewünschte Impulsterm beim Lernen angegeben.

- *Modul-Methoden:*
  - `simpleNet(const size_t dimI, const size_t dimH, const size_t dimO)`
    Dieser Konstruktor erzeugt ein Neuronales Netz mit den übergebenen Dimensionen, stellt Speicher für alle Vektoren und Matrizen bereit und setzt die Gewichtsmatrizen auf einen zufälligen Wert im Intervall [-0.3 … 0.3].
    Vorbedingung: keine
    Nachbedingung: Neuronales Netz erzeugt und die Gewichtsmatrizen mit zufälligen Werten belegt.
  - `simpleNet(const size_t dimI, const size_t dimH, const size_t dimO, const float **ihW, const float **hoW)`
    Dieser Konstruktor erzeugt ein Neuronales Netz mit den übergebenen Dimensionen, stellt Speicher für alle Vektoren und Matrizen bereit und initialisiert die Gewichtsmatrizen mit den übergebenen float-Matrizen.
    Vorbedingung: keine
    Nachbedingung: Neuronales Netz erzeugt und die Gewichtsmatrizen mit den übergebenen Matrizen initialisiert.
  - `simpleNet(const string& filename)`
    Dieser Konstruktor erzeugt ein Neuronales Netz durch Laden aus der übergebenen Datei.
    Vorbedingung: keine
    Nachbedingung: Neuronales Netz wurde aus Datei geladen, bei einem Fehler wird eine ANN_ERROR::file_exception geworfen.
  - `void SaveWeightsToFile(const string& filename)`
    Durch diese Methode wird das aktuelle Neuronale Netz mit seinen Dimensionen und aktuellen Gewichtsmatrizen in der übergebenen Datei gespeichert.
    Vorbedingung: keine
    Nachbedingung: Neuronales Netz wurde in Datei gespeichert, bei einem Fehler wird eine ANN_ERROR::file_exception geworfen.
  - `float GetE()`
    Der quadratische Fehler des letzten Trainings-Laufs durch das Neuronale Netz wird zurückgegeben.
    Vorbedingung: Trainings-Lauf wurde absolviert (mit Train())
    Nachbedingung: Der quadratische Fehler wurde zurückgegeben.
  - `float *GetOutput()`
    Die Methode liefert den m_pOutO-Vektor zurück, welcher die Ausgabe des Netzes gespeichert hat.
    Vorbedingung: Output-Vektor wurde berechnet (mit Run() oder Train()).
    Nachbedingung: m_outO wurde zurückgegeben.
  - `size_t GetInputNeurons()`
    Die Anzahl der Eingabeneuronen wird zurückgegeben.
    Vorbedingung: keine
    Nachbedingung: m_amountI zurückgegeben.

- `size_t GetHiddenNeurons()`
  Die Anzahl der verborgenen Neuronen wird zurückgegeben.
  Vorbedingung: keine
  Nachbedingung: m_amountH zurückgegeben.
- `size_t GetOutputNeurons()`
  Die Anzahl der Ausgabeneuronen wird zurückgegeben.
  Vorbedingung: keine
  Nachbedingung: m_amountO zurückgegeben.
- `void SetErrTolerance(const float tol)`
  Die Fehlertoleranz m_errTol wird auf den übergebenen Wert gesetzt.
  Vorbedingung: keine
  Nachbedingung: m_errTol = tol gesetzt.
- `void SetLearnRate(const float rate)`
  Die Lernrate m_learnRate wird auf den übergebenen Wert gesetzt.
  Vorbedingung: keine
  Nachbedingung: m_learnRate = rate gesetzt.
- `void SetImpuls(const float imp)`
  Der Impulsterm m_impuls wird auf den übergebenen Wert gesetzt.
  Vorbedingung: keine
  Nachbedingung: m_impuls = imp gesetzt.
- `float *Run(const float *input)`
  Das Neuronale Netz wird mit dem entsprechenden Input einmal vorwärts durchlaufen und der dabei berechnete Output wird zurückgegeben.
  Vorbedingung: keine
  Nachbedingung: Der vom Netz zum Input berechnete Output-Vektor wurde zurückgegeben.
- `bool Train(const float *input, const float *pOutput)`
  Es wird ein Trainingslauf durch das Neuronale Netz absolviert. Dabei wird das Netz mit dem übergebenen Input vorwärts durchlaufen und die Abweichung vom tatsächlichen Output zur gewünschten Ausgabe pOutput berechnet. Dann folgt ein Rückwärtsdurchlauf, wobei die Gewichtsmatrizen mittels Backpropagation-Algorithmus angepasst werden, um die Differenz zwischen tatsächlichem und gewünschtem Output zu minimieren.
  Vorbedingung: keine
  Nachbedingung: Trainingslauf durch das Netz wurde absolviert und die Gewichte entsprechend angepasst. Liegt der gemachte Fehler unter m_errTol, so wird true, ansonsten false zurückgegeben.

3.5.1.8. valmatrix<T>:

- *Detaillierte Darstellung:*



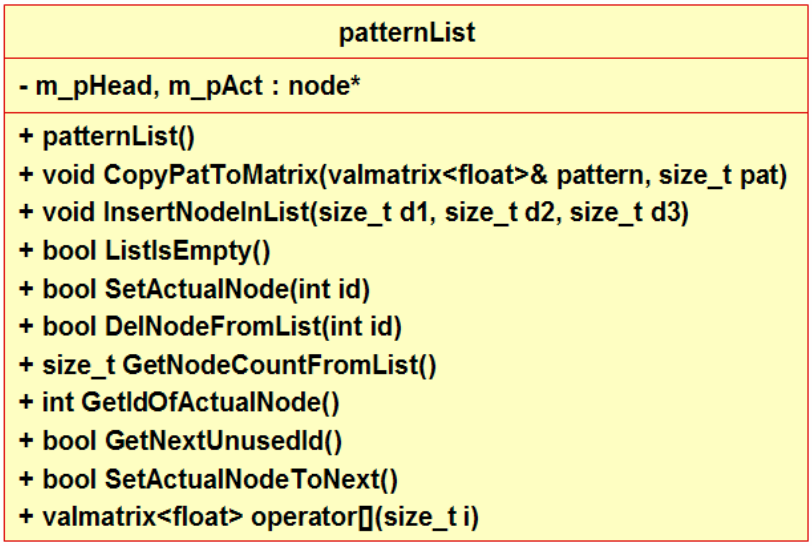
- *Funktionalitätsbeschreibung:*
  Diese Klasse ist die Implementierung einer 2D-Matrix unter Verwendung der C++ Datentyen valarray und slice, welche Bestandteil der Standardbibliothek sind. Mit Hilfe dieser kann eine 2D-Matrix simuliert werden, sodass mit valmatrix ein elegantes objektorientiertes Arbeiten mit einer Klasse für diesen Matrix-Typ möglich ist.

- *Modul-Attribute:*
  - `valarray<T>* m_v`
    Dieses Attribut stellt das valarray dar, welches die Daten aufnimmt, die dann durch slices auf eine 2D-Matrix abgebildet werden.
  - `size_t m_d1, m_d2`
    Diese beiden positiven Ganzzahlen sind die Dimensionen der 2D-Matrix, die bei Erzeugung der Matrix übergeben und dann gespeichert werden.

- *Modul-Methoden:*
  - `valmatrix(size_t y, size_t x)`
    Der Klassenkonstruktor erzeugt eine neue 2D-Matrix mit den Dimensionen y und x.
    Vorbedingung: keine
    Nachbedingung: 2D-Matrix wurde erzeugt und Speicher wurde reserviert.
  - `size_t Size() / size_t Dim1() / size_t Dim2()`
    Diese Methoden geben die Gesamtzahl an Matrixelementen bzw. die Elemente einer Zeile bzw. Spalte zurück.
    Vorbedingung: keine
    Nachbedingung: Der gewünschte Wert wurde zurückgegeben.
  - `slice_iter<T>  operator[](size_t i)`
    Durch Operator-Überladung kann man direkt auf die Elemente der 2D-Matrix mittels Indizierung zugreifen. slice_iter erlaubt ebenfalls eine Indizierung, sodass auf das Element in der Zeile i und der Spalte j einer valmatrix x wie gewohnt mittels x[i][j] zugegriffen werden kann.
    Vorbedingung: keine
    Nachbedingung: Der jeweilige indizierte Slice wurde zurückgegeben.
  - `valarray<T>& Array()`
    Diese Methode gibt das Array zurück, welches als Attribut in der Klasse gespeichert ist (valarray m_v). Dadurch kann die 2D-Matrix ohne Umrechnung auf das 1D-valarray abgebildet werden.
    Vorbedingung: keine
    Nachbedingung: Das valarray, mit welchem die 2D-Matrix simuliert wird, wurde zurückgegeben.

3.5.1.9. patternList:

- *Detaillierte Darstellung:*



- *Funktionalitätsbeschreibung:*
  Diese Komponente stellt eine einfach vorwärts verkettete Liste bereit, welche in jedem Knoten einen kompletten Zeichensatz (also eine Matrix aus Mustern, die in je einer valmatrix gespeichert sind) und eine zum Zeichensatz eindeutig zuordbare ID aufnimmt. Weiterhin werden Methoden bereitgestellt, um eindeutige IDs zu bestimmen, Knoten einzufügen und zu löschen, den aktuell betrachteten Knoten umzusetzen oder auch auf die Matrix des aktuellen Knotens mittels Operator [] zugreifen zu können.

- *Modul-Attribute:*
  - `struct node {`
    `int id;`
    `valmatrix<float>* m_pMatrPattern;`
    `node *next;`
    `};`
    Dieser Struktur definiert einen Knoten der verketteten Liste. Dieser hat eine ID und die zugehörige Matrix, die einen kompletten Zeichensatz aufnehmen kann. Schlussendlich ist wie bei jeder verketteten Liste noch ein Zeiger auf das nächste Listenelement enthalten.
  - `node *m_pHead, *m_pAct`
    Dies sind Eckknoten der verketteten Liste. m_pHead ist der Kopf, m_pAct das aktuelle Element.

- *Modul-Methoden:*
  - `patternList()`
    Der Default-Konstruktor erzeugt eine leere Liste durch Setzen der Eckknoten m_pHead und m_pAct auf NULL.
  - `void CopyPatToMatrix(valmatrix<float>& pattern, size_t pat)`
    Diese Methode kopiert das durch pattern angegebene Muster auf die Position pat in der Zeichensatz-Matrix des aktuellen Knotens.
    Vorbedingung: Zeiger auf aktuellen Knoten m_pAct befindet sich auf einem gültigen Knoten der verketteten Liste.
    Nachbedingung: Das Muster wurde in die Matrix des aktuellen Knotens an Position pat eingefügt.
  - `void InsertNodeInList(size_t d1, size_t d2, size_t d3)`

Mit dieser Methode wird ein neuer Knoten in der Liste erzeugt, wobei die Zeichensatz-Matrix des Knotens die Dimensionen d1, d2, d3 bekommt. Die ID des einzufügenden Knotens wird durch GetNextUnusedId() bestimmt.
Vorbedingung: keine
Nachbedingung: Knoten wurde in Liste eingefügt und Speicher für die Zeichensatz-Matrix des Knotens wurde bereitgestellt. m_pAct wurde auf den eingefügten Knoten gesetzt und m_pHead bei Bedarf umgesetzt.

- `bool ListIsEmpty()`
  Diese Methode gibt true zurück, wenn die Liste leer ist und ansonsten false.
  Vorbedingung: keine
  Nachbedingung: Entscheidung zurückgegeben, ob die Liste leer ist.
- `bool SetActualNode(int id)`
  Der aktuelle Zeiger auf einen Knoten der Liste m_pAct wird auf den Knoten mit der ID id umgesetzt.
  Vorbedingung: keine
  Nachbedingung: m_pAct wurde entsprechend umgesetzt und es wurde true zurückgegeben. Sollte der Knoten nicht in der Liste vorhanden sein, so wird false zurückgegeben und m_pAct wird nicht berührt.
- `bool DelNodeFromList(int id)`
  Der mit id gekennzeichnete Knoten wird aus der verketteten Liste gelöscht.
  Vorbedingung: keine
  Nachbedingung: Der Knoten mit der ID id wurde gelöscht und m_pAct auf den nächsten gültigen Knoten gesetzt. Bei Bedarf wird m_pHead umgesetzt, auf NULL, wenn die Liste leer ist. Im Fehlerfall, also wenn kein Knoten mit der übergegeben ID existiert, wird false zurückgegeben, true sonst.
- `size_t GetNodeCountFromList()`
  Es wird die Anzahl der Knoten in der Liste gezählt und zurückgegeben.
  Vorbedingung: keine
  Nachbedingung: Die Knoten in der Liste wurden gezählt und zurückgegeben.
- `int GetIdOfActualNode()`
  Diese Methode liefert die ID des aktuellen Knotens zurück.
  Vorbedingung: m_pAct zeigt auf einen gültigen node.
  Nachbedingung: ID des aktuellen Knotens zurückgegeben oder -1, wenn die Liste leer ist.
- `int GetNextUnusedId()`
  Diese Methode durchsucht die Liste nach der nächsten freien ID, ausgehend von 0.
  Vorbedingung: keine
  Nachbedingung: Nächste freie ID wurde zurückgegeben.
- `bool SetActualNodeToNext()`
  m_pAct, also der Zeiger auf den aktuellen Knoten wird auf das nächste Listenelement gesetzt.
  Vorbedingung: m_pAct zeigt auf einen gültigen node.
  Nachbedingung: m_pAct zeigt auf den nächsten Knoten der Liste. Sollte es keinen nächsten Knoten geben, so wird false zurückgegeben, true sonst.
- `valmatrix<float> operator[](size_t i)`
  Bei Indizierung einer Instanz von patternList wird durch die Operatorüberladung das i. Muster der Zeichensatz-Matrix des *aktuellen* Knotens zurückgegeben. Das macht das Arbeiten mit den Zeichensätzen der verketteten Liste komfortabel.
  Vorbedingung: m_pAct zeigt auf einen gültigen node.
  Nachbedingung: Das Muster i des Zeichensatzes vom aktuellen Knoten wurde zurückgegeben oder NULL, wenn kein aktueller Knoten existiert.

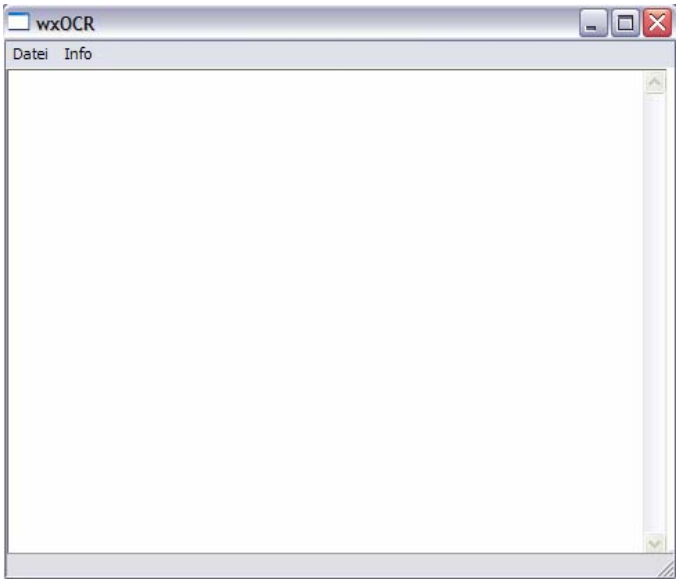
3.5.2. Schicht 2: Benutzungsoberfläche:

3.5.2.1. wxOCRFrame:

- *Detaillierte Darstellung:*

wxOCRFrame
- m_pMenuBar : wxMenuBar* - m_pFileMenu, m_pInfoMenu : wxMenu* - m_pOutText : wxTextCtrl* - m_pRecogMain : wxOCRRecogMain*
+ wxOCRFrame(const wxChar* title, int x, int y, int w, int h) + void OnMenuFileLoadImg(wxCommandEvent& event) + void OnMenuFileSaveTextToFile(wxCommandEvent& event) + void OnMenuFileQuit(wxCommandEvent& event) + void OnMenuInfoAbout(wxCommandEvent& event)

- *So sieht's aus:*



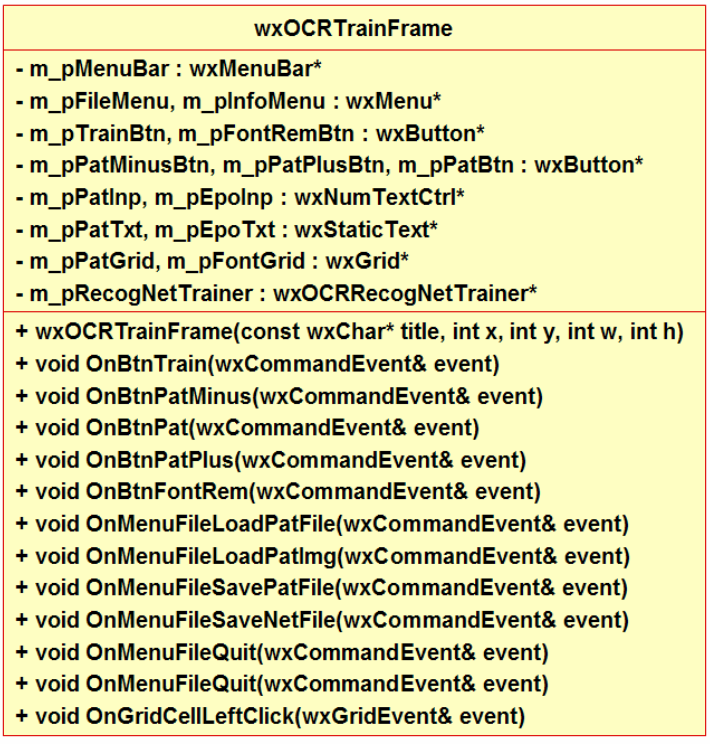
- *Funktionalitätsbeschreibung:*
  Diese Klasse ist von der wxWidgets-Klasse wxFrame abgeleitet und dient der Darstellung des Hauptfensters von wxOCR mit seinen Komponenten. Das Hauptfenster soll nur aus einem Textfeld bestehen, in welches der erkannte Text ausgegeben wird. Hinzu kommt die Menüleiste, weitere Items sollen aber nicht vorhanden sein und sind auch nicht nötig.

- *Modul-Attribute:*
  - `wxMenuBar *m_pMenuBar`
    Dieses Attribut stellt die Menübar dar, auf der die beiden Menüs „Datei“ und „Info“ angebracht sind.
  - `wxMenu *m_pFileMenu, *m_pInfoMenu`
    Dies sind Attribute für die beiden Menüs „Datei“ und „Info“.
  - `wxTextCtrl *m_pOutText`
    Dies ist das Textfeld, in das der erkannte Text ausgegeben wird.
  - `wxOCRRecogMain *m_pRecogMain`
    Dies ist eine Instanz der Funktionalitätskomponente von wxOCR, über welche das Neuronale Netz und ein Bild geladen werden können und welche aus einem Bild erkannten Text zurückgeben kann.

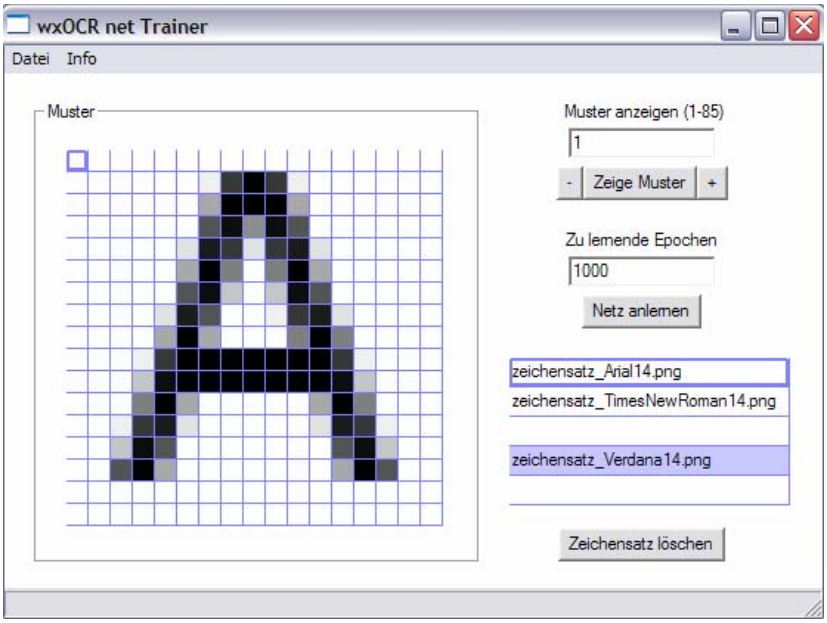
- *Modul-Methoden:*
  - `wxOCRFrame(const wxChar* title, int x, int y, int w, int h)`
    Der Konstruktor erzeugt das Frame mit dem angegebenen Titel, der x- und y-Position und der Breite und Höhe. Dazu wird zunächst der Konstruktor von wxFrame() aufgerufen, dann werden die gewünschten Komponenten und das Fensterlayout erzeugt.
    Vorbedingung: keine
    Nachbedingung: Fenster wurde erzeugt und angezeigt.
  - `void OnMenuFileLoadImg(wxCommandEvent& event)`
    Wird ausgeführt, wenn der Benutzer aus dem Datei-Menü die Option „Bild laden“ wählt. In diesem Fall wird das gewählte Bild geladen und von m_pRecogMain der enthaltene Text erkannt.
    Vorbedingung: keine
    Nachbedingung: Bild wurde geladen und der enthaltene Text in m_pOutText ausgegeben.
  - `void OnMenuFileSaveTextToFile(wxCommandEvent& event)`
    Wählt der Benutzer Datei->“Text in Datei speichern“, so wird diese Methode ausgeführt.
    Vorbedingung: Text wurde aus Bild erkannt und in m_pOutText ausgegeben.
    Nachbedingung: In m_pOutText enthaltener Text wurde in der vom Benutzer gewählten Datei gespeichert.
  - `void OnMenuFileQuit(wxCommandEvent& event)`
    Das Programm wird verlassen.
    Vorbedingung: keine
    Nachbedingung: Programm beendet.
  - `void OnMenuInfoAbout(wxCommandEvent& event)`
    Wird ausgeführt, wenn der Benutzer den Menüpunkt zur Informationsanzeige über das Programm auswählt.
    Vorbedingung: keine
    Nachbedingung: Informationsdialog wurde präsentiert.

3.5.2.2. wxOCRTrainFrame:

- *Detaillierte Darstellung:*



- *So sieht's aus:*



- *Funktionalitätsbeschreibung:*
  wxOCRTrainFrame (von wxFrame abgeleitet) ist das Pendant im wxOCR net Trainer zu wxOCRFrame im Hauptprogramm. Es wird ein Fenster erzeugt mit all seinen darzustellenden Komponenten und dem gewünschten Layout. In diesem kann der Benutzer nun durch Interaktion Funktionen ausführen, im net Trainer speziell Zeichensätze laden und das Neuronale Netz damit anlernen.

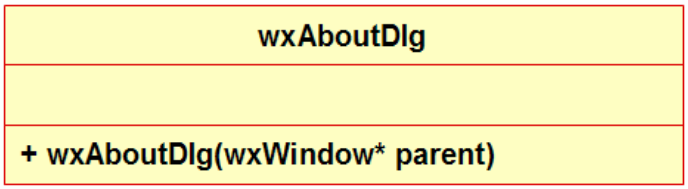
- *Modul-Attribute:*
  - `wxMenuBar *m_pMenuBar`
    Die Menübar des Fensters.
  - `wxMenu *m_pFileMenu, *m_pInfoMenu`
    Die Menüs, welche in der Menübar angezeigt werden.
  - `wxButton *m_pTrainBtn, *m_pPatMinusBtn, *m_pPatBtn, *m_pPatPlusBtn, *m_pFontRemBtn`
    Buttons, welche benötigt werden. Mit m_pTrainBtn kann das Neuronale Netz trainiert werden, mit m_pPatBtn wird ein geladenes Muster angezeigt, mit m_pPatMinusBtn, m_pPatPlusBtn werden das vorige bzw. nächste Muster angezeigt und mit m_pFontRemBtn kann ein geladener Zeichensatz wieder gelöscht werden.
  - `wxNumTextCtrl *m_pPatInp, *m_pEpoInp`
    Numerische Texteingabe-Felder. m_pPatInp erlaubt die Eingabe eines anzuzeigenden Musters, m_pEpoInp das Lernen des Neuronalen Netzes mit einer angegebenen Epochenzahl.
  - `wxStaticText *m_pPatTxt, *m_pEpoTxt`
    Statische Textfelder zur Ausgabe von Text.
  - `wxGrid *m_pPatGrid, *m_pFontGrid`
    Dies sind Tabellen (Grids), die im Fenster angezeigt werden sollen. m_pPatGrid kann ein geladenes Muster in entsprechender Größe anzeigen, m_pFontGrid zeigt die Namen aller geladenen Zeichensätze an.
  - `wxOCRRecogNetTrainer *m_pRecogNetTrainer`
    Die Funktionalitätskomponente des net Trainers, über welche z.B. Zeichensätze geladen, das Netz angelernt und in einer Datei gespeichert werden kann.

- *Modul-Methoden:*
  - `wxOCRTrainFrame(wxChar* title, int x, int y, int w, int h)`
    Der Konstruktor erzeugt das Frame mit dem angegebenen Titel, der x- und y-Position und der Breite und Höhe. Dazu wird zunächst der Konstruktor von wxFrame() aufgerufen, dann werden die gewünschten Komponenten und das Fensterlayout erzeugt.
    Vorbedingung: keine
    Nachbedingung: Fenster wurde erzeugt und angezeigt.
  - `void OnBtnTrain(wxCommandEvent& event)`
    Wird ausgeführt, wenn der Benutzer den Button zum Trainieren des Netzes betätigt.
    Vorbedingung: Es wurde bereits ein zu lernender Zeichensatz geladen.
    Nachbedingung: Neuronales Netz wurde mit der gewünschten Epochenzahl und den im m_pFontGrid vorhandenen Zeichensätzen angelernt, der Menüpunkt zum Speichern des Neuronalen Netzes wurde aktiviert.
  - `void OnBtnPatMinus(wxCommandEvent& event)`
    Wird ausgeführt, wenn das vorige Muster in m_pPatGrid ausgegeben werden soll.
    Vorbedingung: Es wurde bereits ein Zeichensatz geladen.
    Nachbedingung: Muster wurde in m_pPatGrid ausgegeben.
  - `void OnBtnPat(wxCommandEvent& event)`
    Wird ausgeführt, wenn das nächste Muster in m_pPatGrid ausgegeben werden soll.
    Vorbedingung: Es wurde bereits ein Zeichensatz geladen.
    Nachbedingung: Muster wurde in m_pPatGrid ausgegeben.
  - `void OnBtnPatPlus(wxCommandEvent& event)`
    Wird ausgeführt, wenn das Muster mit der gewählten Nummer in m_pPatGrid ausgegeben werden soll.
    Vorbedingung: Es wurde bereits ein Zeichensatz geladen.
    Nachbedingung: Muster wurde in m_pPatGrid ausgegeben.
  - `void OnBtnFontRem(wxCommandEvent& event)`
    Wird ausgeführt, wenn der Benutzer den Button zum Löschen eines gewählten Zeichensatzes gewählt hat.
    Vorbedingung: Es wurde bereits ein Zeichensatz geladen.
    Nachbedingung: Der gewählte Zeichensatz wurde gelöscht und aus m_pFontGrid entfernt, dafür wurde ein anderer geladener Zeichensatz gewählt.
  - `void OnMenuFileLoadPatFile(wxCommandEvent& event)`
    Wird ausgeführt, wenn der Benutzer den Menüpunkt zum Laden eines Musters aus einer Datei gewählt hat.
    Vorbedingung: keine
    Nachbedingung: Muster wurden aus Datei geladen oder ein Fehler ausgegeben, wenn die Muster nicht gelesen werden konnten. Items zur Auswahl von Mustern und zum Anlernen des Neuronalen Netzes wurden aktiviert.
  - `void OnMenuFileLoadPatImg(wxCommandEvent& event)`
    Wird ausgeführt, wenn der Benutzer den Menüpunkt zum Laden eines Musters aus einem Bild gewählt hat.
    Vorbedingung: keine
    Nachbedingung: Muster wurden aus Bild extrahiert und geladen oder ein Fehler ausgegeben, wenn die Muster nicht gelesen werden konnten. Items zur Auswahl von Mustern und zum Anlernen des Neuronalen Netzes wurden aktiviert.
  - `void OnMenuFileSavePatFile(wxCommandEvent& event)`
    Wird ausgeführt, wenn der Benutzer den Menüpunkt zum Speichern der Muster in einer Datei gewählt hat.
    Vorbedingung: Zeichensatz wurde aus Bild/Datei eingeladen.
    Nachbedingung: Der aktuell im m_pFontGrid gewählte Zeichensatz wurde in der vom Benutzer gewählten Datei gespeichert oder es wurde bei einem Fehler eine Fehlermeldung ausgegeben.

- `void OnMenuFileSaveNetFile(wxCommandEvent& event)`
  Wird ausgeführt, wenn der Benutzer das angelernte Neuronale Netz in einer gewählten Datei speichern will.
  Vorbedingung: Zeichensatz wurde geladen und Neuronales Netz damit angelernt.
  Nachbedingung: Neuronales Netz wurde in der gewählten Datei gespeichert oder eine Fehlermeldung ausgegeben, wenn etwas schief ging.
- `void OnMenuFileQuit(wxCommandEvent& event)`
  Programm wird verlassen.
  Vorbedingung: keine
  Nachbedingung: Programm beendet.
- `void OnMenuInfoAbout(wxCommandEvent& event)`
  Wird ausgeführt, wenn der Benutzer den Menüpunkt zur Informationsanzeige über das Programm auswählt.
  Vorbedingung: keine
  Nachbedingung: Informationsdialog wurde präsentiert.
- `void OnGridCellLeftClick(wxGridEvent& event)`
  Wird ausgeführt, wenn der Benutzer auf eine der beiden Tabellen im Fenster klickt.
  Vorbedingung: Zeichensatz wurde aus Datei/Bild geladen.
  Nachbedingung: Wurde in m_pPatGrid geklickt, so passiert nichts. Klickt der Nutzer in m_pFontGrid, wählt also einen anderen Zeichensatz aus, so wird dieser gesetzt und im m_pPatGrid angezeigt.

3.5.2.3. wxAboutDlg:

- *Detaillierte Darstellung:*



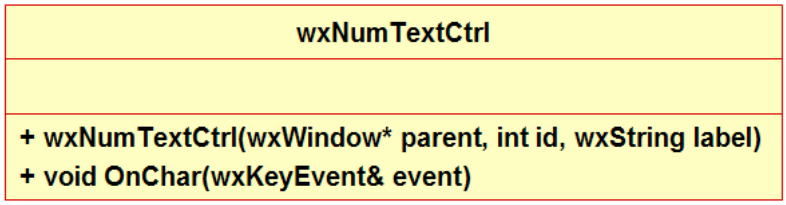
- *Funktionalitätsbeschreibung:*
  Zweck dieser Klasse ist lediglich, einen Dialog mit Informationen zu dem wxOCR-Projekt anzuzeigen.

- *Modul-Attribute:*
  - Keine

- *Modul-Methoden:*
  - `wxAboutDlg(wxWindow* parent)`
    Konstruktor für den Dialog, der ein neues Dialogfenster erzeugt und dort den Info-Text zum Projekt anzeigt.
    Vorbedingung: es existiert bereits ein Frame oder ein Objekt vom Typ wxWindow.
    Nachbedingung: Info-Dialog wurde angezeigt.

3.5.2.4. wxNumTextCtrl:

- *Detaillierte Darstellung:*



- *Funktionalitätsbeschreibung:*
  Diese Klasse erbt von der wxWidgets-Komponente wxTextCtrl und ist dafür zuständig, eine Texteingabe-Komponente zu erzeugen, die nur numerische ganzzahlige Eingaben und nur solche zwischen 0 und 10000 akzeptiert. Damit kann z.B. die Eingabe der Lernepochen für das Neuronale Netz realisiert werden.

- *Modul-Attribute:*
  - Keine

- *Modul-Methoden:*
  - `wxNumTextCtrl(wxWindow* parent, int id, wxString label)`
    Der Konstruktor erzeugt die numerische Texteingabe-Komponente durch Aufruf des Konstruktors der Basisklasse wxTextCtrl.
    Vorbedingung: keine
    Nachbedingung: numerische Texteingabe-Komponente erzeugt.
  - `void OnChar(wxKeyEvent& event)`
    Wird aufgerufen, wenn ein Zeichen in das Textfeld eingegeben wurde. Handelt es sich um eine Ziffer, so wird sie zur Ausgabe hinzugefügt, sonst nicht.
    Vorbedingung: keine
    Nachbedingung: Textfeld wurde ergänzt, wenn das eingegebene Zeichen eine Ziffer war.

3.5.3. Unterschiede zwischen den Teilapplikationen:

Zu wxOCR (main) gehören dediziert die Module wxOCR_main, wxOCRFrame und wxOCRRecogMain. wxOCR net Trainer beinhaltet äquivalent dazu wxOCR_netTrainer, wxOCRTrainFrame und wxOCRRecogNetTrainer.
Alle anderen Module werden von beiden Programmen gemeinsam benutzt. Grund hierfür ist die Algorithmik des Neuronalen Netzes sowie der Bildverarbeitungskomponente, die von beiden Teilapplikationen gleichermaßen genutzt werden muss. So muss der netTrainer, der eigentlich nur für das Anlernen des Neuronalen Netzes zuständig ist, auch Bilder verarbeiten können, da Zeichensätze auch einfach aus Bildern geladen werden sollen.

3.5.4. Verdeutlichung der Bildverarbeitungs-Anpassbarkeit:

Wie schon mehrfach erwähnt, kann ein Entwickler leicht das Bildverarbeitungsmodul austauschen, indem er seine eigene Klasse zur Bildverarbeitung und Muster-Extrahierung aus einem Bild schreibt, welche von wxOCRImageProc abgeleitet ist und diese implementiert.

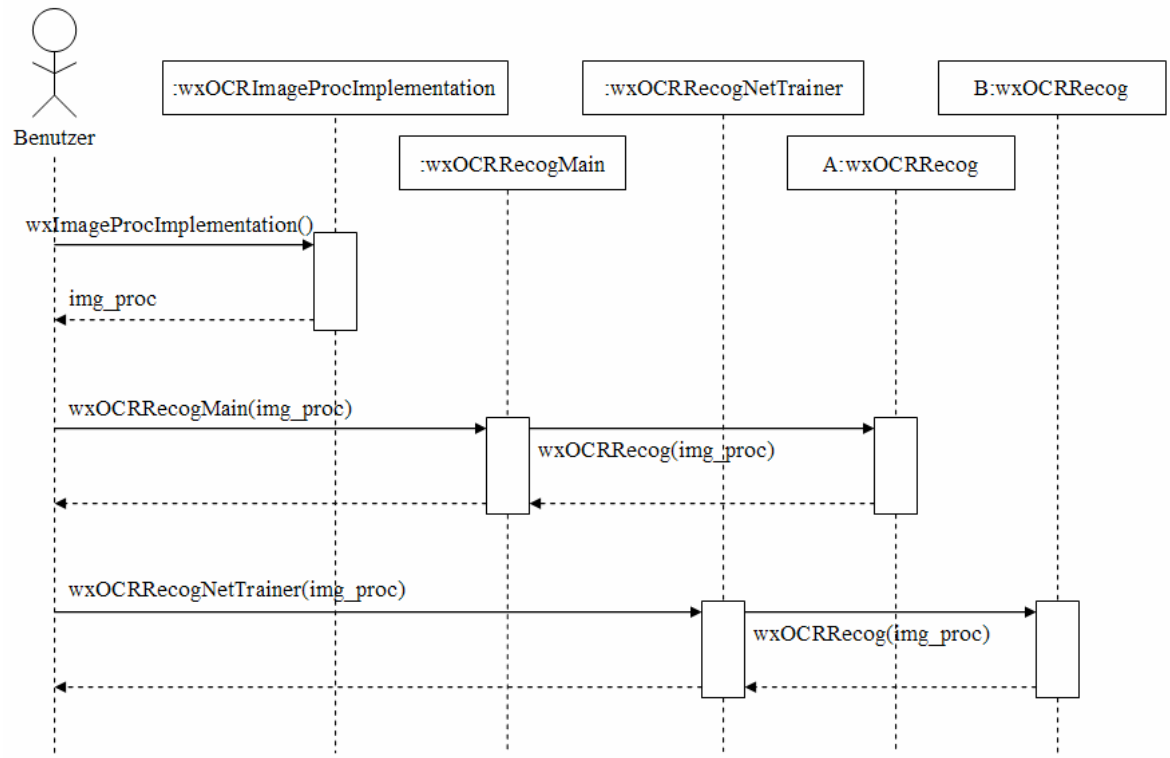
Diese benutzerdefinierte Klasse kann dann sehr einfach in das bestehende Programm eingebunden werden, indem je nach Teilapplikation entweder wxOCRRecogMain() oder wxOCRRecogNetTrainer() mit einer konkreten Instanz der selbst geschriebenen Klasse parametrisiert werden.

Hat ein Benutzer beispielsweise eine Klasse wxOCRImageProcImplementation, so muss er zur Einbindung in das Hauptprogramm nur schreiben:

```
wxOCRRecogMain *m_pRecogMain = new wxOCRRecogMain(
                                   new wxOCRImageProcImplementation)
```

und hat damit seine eigene Bildverarbeitungsklasse in das Programm integriert.

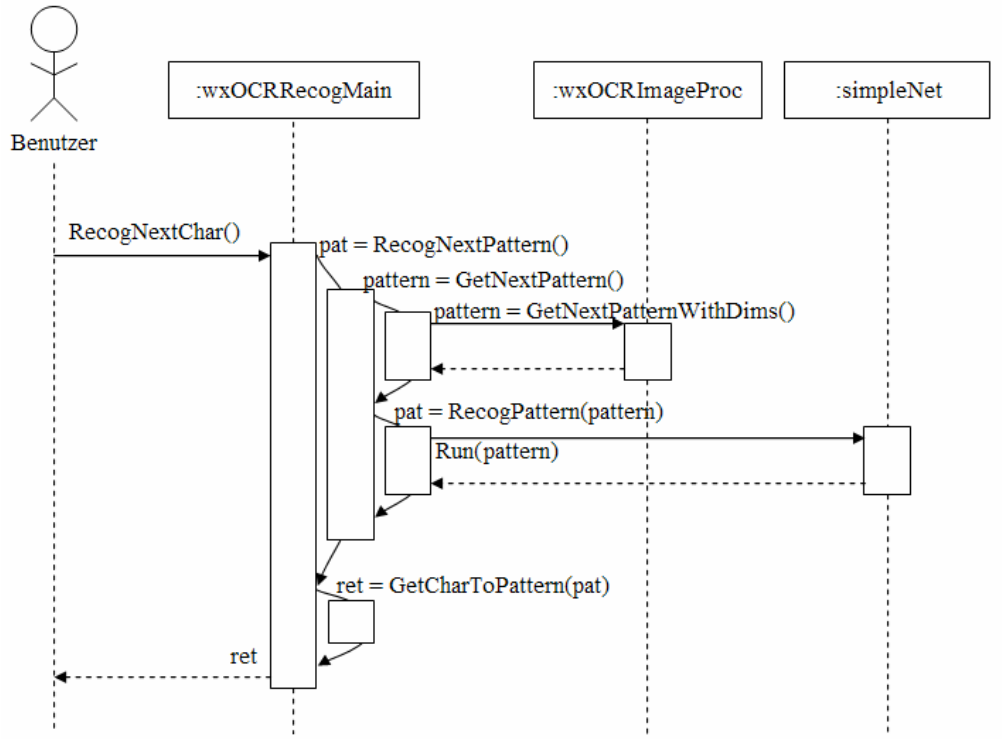
Folgendes Sequenzdiagramm verdeutlicht dieses Prinzip:



3.5.5. Verdeutlichung der Erkennung eines Zeichens aus einem Bild:

Zeichen werden aus einem Bild erkannt, indem sequentiell die Methode GetNextPatternWithDims() einer Implementierung der Klasse wxOCRImageProc aufgerufen wird und man das Neuronale Netz zur Erkennung des Zeichens mit dem so extrahierten Muster füttert.

Ein Benutzer der Klasse wxOCRRecogMain des Hauptprogramms beispielsweise kann deren Methode RecogNextChar() aufrufen, die genau das realisiert, wie das folgende Sequenzdiagramm veranschaulicht:



4. Arbeits-Aufteilung

	Matthias Jauernig	Michael Lahl
Programmierung	• Schicht 1: Eigentliche Anwendung	• Schicht 2: Benutzungsoberfläche • Unterstützend bei Schicht 1
Dokumentation	• Quelltextkommentierung • Changelog	• Endbenutzerdokumentation • Entwicklerdokumentation
Organisation	• Administrator der sourceforge.net-Projektseite	• CVS-Verwalter bei sourceforge.net • Gestaltung der Webseite zum Projekt auf sourceforge.net

Die Aufteilung der Programmierarbeit in voneinander unabhängige Teile wird hier durch die 2-Schichten-Architektur unterstützt, welche die Benutzungsoberfläche isoliert von der eigentlichen Anwendung. Über wenige Schnittstellenfunktionen nutzt Schicht 2 die Funktionalität der Schicht 1, in der Gegenrichtung findet keine Nutzung statt…

Begründung der Arbeitsverteilung:

- Michael Lahl: Wissen bei der Grafikprogrammierung mit wxWidgets und Erfahrungen im Erstellen von Dokumentationen und Webseiten sowie der Verwaltung von CVS-Archiven.
- Matthias Jauernig: Wissen und Interesse bei der Programmierung von Neuronalen Netzen und in der Algorithmen- und oo-Programmierung. Erfahrungen bei der Web-Administration und im Projekt-Management.

Zunächst wird der wxOCR net Trainer implementiert werden. Aufgrund der gemeinsamen Nutzung der wichtigsten Algorithmen vom Hauptprogramm sowie der netTrainer-Applikation wird es dann ein Einfaches sein, das Hauptprogramm zu implementieren.

5. Style-Guide-Vorgaben

5.1. Textformatierung:

- *Einrückung:* 4 Leerzeichen, anzuwenden bei Zeilenumbrüchen, innerhalb von Blöcken, bei if-Anweisungen und Schleifen mit nur einer Anweisung sowie nach speziellen Schlüsselwörtern wie private/public/…
- *Leerzeilen:* mindestens zwischen jeder Funktionsdefinition, die aus mehr als einer Zeile besteht und vor bzw. nach im Kontext zusammen gehörigen Codeteilen innerhalb von Funktionen.
- *Zeilenumbrüche:* Beim Erreichen der maximalen Anzahl an Zeichen pro Zeile.
- *Leerzeichen:* Mindestens zwischen den am höchsten priorisierten Operatoren einer Anweisungszeile, nicht bei unären Operatoren wie [ ] oder ++. Auch zur Trennung von einzelnen Übergabewerten bei Funktionsaufrufen.
  Z.B.: „int i = 0; i = f(1, 2, 3);“

5.2. Namensgebung:

- Allgemein: aussagekräftige, aber nicht zu lange Namen in Anlehnung an die englische Sprache.
- *Klassen:* Beginn mit Kleinbuchstaben, Trennung einzelner Wörter im Klassennamen durch Beginn des nächsten Wortes mit einem Großbuchstaben, z.B. „class simpleNet“.
- *Methoden:* Beginnen mit einem Großbuchstaben, einzelne Wörter im Methodennamen sind wie bei Klassen durch Großbuchstaben getrennt, z.B. „void MyFunctionRocks();“.
- *Klassenattribute:* Beginnen mit m_ gefolgt von dem Attributnamen, z.B. „int m_attr“. Ansonsten äquivalent zu der Namensgebung von Variablen.
- *Konstanten:* mit #define deklarierte symbolische Konstanten sind komplett mit Großbuchstaben zu schreiben, z.B. „#define TEN 10“.
- *Variablen:* Keine nähere Typkennzeichnung im Variablennamen, Namensgebung wie bei Klassen, nur Zeiger werden durch ein zusätzliches p am Variablenanfang gekennzeichnet, z.B. „int myLocalVar; float** pDim2Array“.

5.3. Obergrenzen:

- Max. Zeichen pro Zeile: 100 Zeichen.
- Keine Festlegung bezüglich der max. Zeilenanzahl pro Methode/Funktion.

5.4. Kommentare / Dokumentation:

- Entwickler- und Endbenutzerdokumentation ausgelagert in separate HTML-Dateien.
- *Kommentare am Anfang einer Quellcode-Datei:* zunächst steht in jeder Datei ein Hinweis auf die GPL als einheitlicher mehrzeiliger Kommentar. Weiterhin ist ein mehrzeiliger Kommentar vorgesehen, der mit einem 78 Zeichen langen /* ###...### */ beginnt und endet. Dieser soll eine möglichst knappe Beschreibung der Funktionalität liefern, welche in der Datei realisiert wird.
- *Kommentare vor Funktionen/Methoden:* /**/-Kommentare, die mit --- beginnen und eine kurze Methodenbeschreibung enthalten. Der Kommentar wird dann soweit mit "-" aufgefüllt, bis eine einheitliche Länge von 78 Zeichen erreicht wird. Die Kommentare können auch mehrzeilig sein.
- *Kommentare in Funktionen/Methoden:* //-Kommentare, die vor einem zu kommentierenden Block stehen sollten, sich bei Bedarf aber auch hinter einer zu kommentierenden Codezeile befinden können. Können sich auch über mehrere Zeilen erstrecken.

5.5. Eine typische Quelltextdatei:

Diese könnte wie folgt aussehen (nur zur Verdeutlichung der einiger Style Guides):

```
/************************************************************************
 *   Copyright (C) 2005 by Matthias Jauernig, Michael Lahl               *
 *   rtc@linux-related.de, dj-mayday@web.de                              *
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                 *
 ************************************************************************/

/* ########################################################################## */
/* # diese Datei definiert die Klasse testClass, um Vergleichs-          # */
/* # operationen und Methoden für Ganzzahlen zur Verfügung zu stellen    # */
/* ########################################################################## */


/* --- Finden des Maximums von 3 Integern ------------------------------- */
int testClass::MaxOfThree(const int i, const int j, const int k){
    int max = 0;
    m_i = i;
    m_j = j;
    m_k = k;

    // Maximum der 3 Attribute ermitteln und zurückgeben
    if(m_i < m_j){
        if(m_j < m_k)
            max = m_k;
        else
            max = m_j;
    }
    else{
        if(m_i < m_k)
            max = m_k;
        else
            max = m_i;
    }

    return max;
}
```