

Dokumentation zum Beleg im Fach Neuronale Netze

Auffinden von TV Logos

Andreas Schindler
htw8859

Torsten Schöps
htw8849

Stand: 9. Oktober 2000

1. Aufgabenstellung

Für den Beleg im Fach Neuronale Netze haben wir uns für folgendes Thema entschieden:

Die exemplarische Erkennung von TV Logos basierend auf Neuronalen Netzen mit Hilfe der Angabe des TV-Kanals.

Zur Lösung dieser Aufgabenstellung kommt die **SNNS** Software (Stuttgart Neural Network Simulator) und die **JPEG** Library der Independent JPEG Group (Public GNU License) zum Einsatz.

2. Fernsehlogoerkennung - eine Vorbetrachtung

Die Aufgabenstellung hat zum Ziel, eine Bildererkennung an realen Fernsehbildern vorzunehmen. Sie hat jedoch nicht zum Ziel, die Echtzeitproblematik des "Grabbing" der TV-Bilder zu realisieren.

Demnach stützt sich die Software auf bereits abgespeicherte Bilder im JPEG Format. Diese Bilder wurden vorher über eine PCI-TV Karte (Typ: BT848 Chipsatz) unter Windows 98 abgespeichert. Da es sich bei dem zur Bildererkennung relevanten Bereich nur um einen Teilausschnitt aus dem Gesamtbild handelt, beinhalten alle abgespeicherten JPEG Bilder nur den extrahierten Teilbereich. Die Größe des Teilbereiches wurde zu Beginn des Projektes stochastisch ermittelt. Dabei stellte sich heraus, daß heut zu Tage prinzipiell vier Arten von Fernsehlogos existieren:

- Einfarbige Logos - z.B. WDR, VH-1, N3
- Mehrfarbige Logos - z.B. Super RTL, Eurosport, CNN
- Transparente Logos (ein- und mehrfarbig) - z.B. RTL, ARD, VIVA2
- Animierte Logos (teilweise auch transparent) - z.B. MTV, SAT1, RTL2

Weiterhin fanden wir heraus, daß sich der Teilausschnitt abhängig vom TV-Sender, in der linken oberen, rechten oberen oder linken unteren Ecke in einer Matrix von 200x150 Bildpunkten befindet. Um aus Performancegründen den späteren Aufwand bei dem Wichten¹ und Arbeiten mit den Netzen zu minimieren, wurde die Matrix auf eine Größe von 100x75 Bildpunkte skaliert.

Aus der obigen Aufzählung der Arten von Logos ergibt sich eine zunehmende Komplexität im Erkennungsaufwand. Unser erster Testkandidat war daher

¹Einstellen der Kantengewichte des neuronalen Netzes

das Logo von VH-1, bedingt durch die Größe des Logos im Vergleich mit anderen TV Logos. Basierend auf den Erfahrungen an dem VH-1 Logo wurde später eine Erkennung des mehrfarbigen Super RTL Logos implementiert. Zur Verbesserung der resultierenden Leistungsfähigkeit des Netzes wurden weiterhin die JPEG Grafikdaten basierend auf einer 24 Bit Farbpalette, wovon nur 15 Bit (Standard PAL Signal) belegt sind, auf eine 4 Bit Farbpalette normiert und dem Netz präsentiert. Diese normierten Grafikdaten findet man in physischer Form nur in den Patternfiles zum Trainieren des Netzes.

3. Netzaufbau

3.1. Analyse der Netzstruktur

Schon zu Beginn der Entwicklung der Netzstruktur stand die Anzahl der Input- und Outputneuronen fest. Die Größe der Inputschicht ist dabei gleich der Größe der Matrix, die das TV Logo einnimmt. Die Outputschicht enthält ein Neuron, welches die prozentuale Erkennung des Logos aus der Inputschicht widerspiegelt. Was jedoch nicht feststand, war die Anzahl und Größe der Hiddenlayer sowie die Verbindung zwischen den Neuronen.

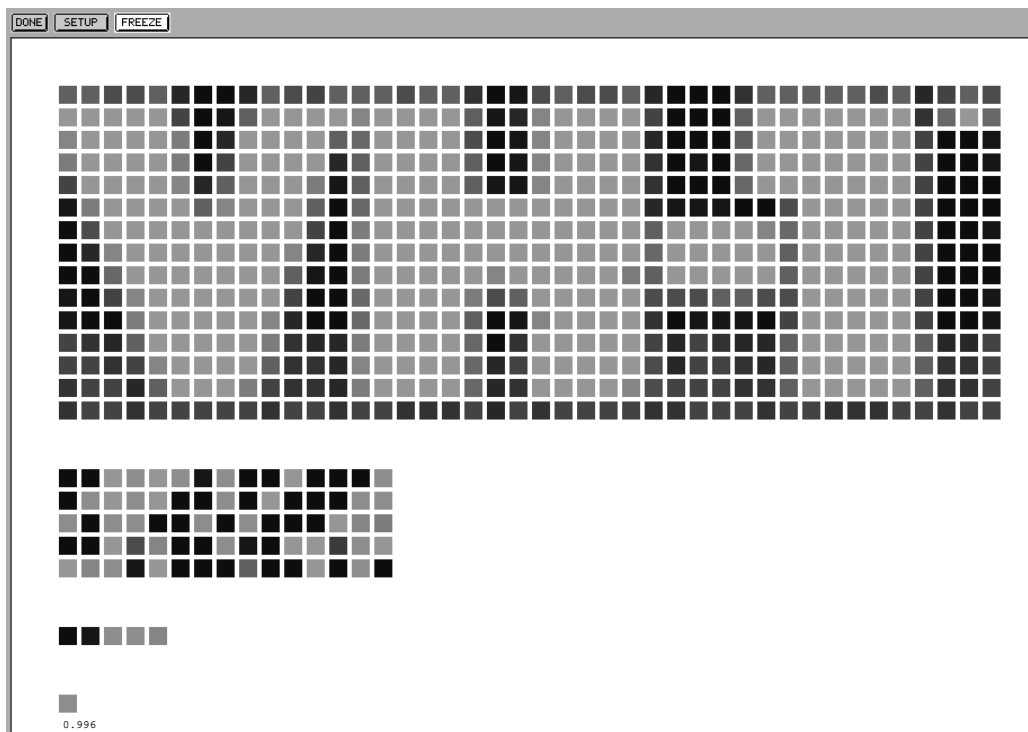


Abb.1 Struktur des Neuronalen Netzes für VH-1 Logo

Um die größtmögliche Anzahl an Mustern dem Netz zu lernen, haben wir uns für eine "Full Connection"-Netzstruktur² entschieden. Bei verschiedenen Testläufen kristallisierte sich folgende Struktur heraus. Die erste Hidden-Schicht hat ein Drittel Größe der Kantenlängen der Inputschicht. Die zweite Hidden-schicht ist ein Vektors und errechnet sich aus der größeren Kantenlänge der ersten Hiddenschicht dividiert durch den Wert drei.

Somit ergeben sich für unsere Testkandidaten diese Größen:

VH-1

Inputschicht:	42 x 15 Neuronen
1. Hiddenschicht:	15 x 5 Neuronen
2. Hiddenschicht:	5 x 1 Neuronen
Outputschicht:	1 Neuron

Super RTL

Inputschicht:	58 x 12 Neuronen
1. Hiddenschicht:	20 x 4 Neuronen
2. Hiddenschicht:	7 x 1 Neuronen
Outputschicht:	1 Neuron

3.2. Erstellen von SNNS Pattern Dateien

Beim Erstellen von Pattern Dateien muß auf eine gute Auswahl der Lernmuster geachtet werden. Als erfolgreiche Strategie stellte sich eine große Anzahl an ähnlichen Mustern heraus, die mit gelegentlichen "Ausreißern" oder "Extremfällen" angereichert ist. Es ist jedoch zu beachten, daß sich die Muster nicht gegenseitig widersprechen. Weiterhin sollten dem Netz positive als auch negative Beispiele gezeigt werden, wobei das Verhältnis von positiven zu negativen Mustern idealerweise 2:1 ist (Erfahrungswerte).

Die Strategie, dem Netz das Logo als eine Reihe von kontinuierlichen Fragmenten des gesamten Logos zu lernen, erwies sich als wenig erfolgreich, da das Netz ähnliche Fragmente in Bildern ohne Logo fälschlicherweise als Positive, also Bild mit Logo, erkannte. Dies hätte zur Folge, daß jedes erkannte Fragment einem weiteren Netz als möglicher Logobeginn präsentiert werden müßte, was eine drastische Erhöhung des Rechen- und Erkennungsaufwandes bedeutet. Die Erkennung eines einzigen Bildes könnte bei heutiger Rechenleistung (Pentium III 450 Mhz) bis zu einer Stunden dauern.

Wir haben uns deshalb zur erst genannten Variante entschlossen. Die Erkennung des Logos erfolgt hier durch eine Schleife, die sämtliche Positionen

²jedes Neuron der Ausgangsschicht ist mit jedem Neuron der Folgeschicht verbunden

der Inputmatrix im Bild (100 x 75 Matrix) durchläuft. Die Rechenzeit wird damit ins erträgliche Maß reduziert. Wichtig ist dabei, daß dem Netz nur Muster gelernt werden, die das Logo annähernd vollständig zeigen, da das Netz eine scharfe Grenze zwischen Positiv- und Negativergebnis ziehen muß. Diese Variante ist entsprechend unserer gewählten Netzstruktur als sehr befriedigend anzusehen. Doch mit einer veränderten Netzstruktur bzw. einer Kombination aus verschiedenen Netzstrukturen sind sicherlich bessere Ergebnisse möglich. Die oben genannten Strategien zum Aufbau eines Patternfiles müßten dafür aber neu überdacht werden.

3.3. Trainieren des Netzes

Das SNNS bringt viele verschiedene Lern- und Aktivierungsfunktionen für den Anwender mit. Für unsere gewählte Netzstruktur sind verschiedene Aktivierungsfunktionen möglich. Diese bedingen jedoch einen unterschiedlichen Wertebereich der Inputschicht. Entsprechend müssen die Patterndateien abgestimmt sein.

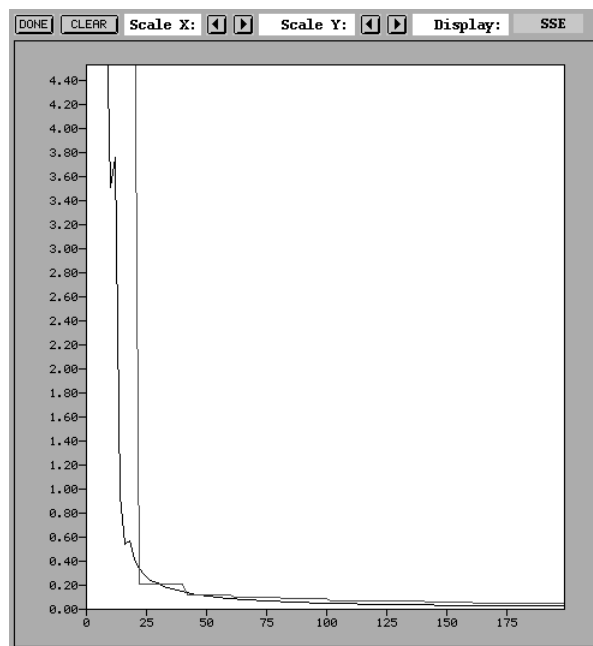


Abb.2 Lerngraph eines Netzes mit Act_Logistic Aktivierungsfunktion und Standard_Backpropagation als Lernfunktion

Dabei stellte sich heraus, daß bei den meisten Aktivierungsfunktionen der Bereich im Intervall $[-1;1]$ bzw. $[0;1]$ optimal ist. Während der Entwicklung

der oben genannten Netzstruktur wurden die Tests ausschließlich mit der logarithmischen Aktivierungsfunktion durchgeführt. Nachdem die endgültige Netzstruktur ermittelt war, trainierten wir dieses Netz mit folgenden Parametern am erfolgreichsten:

Aktivierungsfunktion:	Act_Logistic
Lernfunktion:	Std_Backpropagation
Update Funktion:	Topological_Order
Initialisierung:	Randomize_Weights

Außerdem trainierten wir das Netz mit der Aktivierungsfunktion Act_TanH. Diese arbeitet im Gegensatz zur Logistischen Aktivierungsfunktion (Intervall $[0;1]$) im Intervall $[-1;1]$, brachte jedoch keine befriedigenden Ergebnisse. Auch die Verwendung anderer Lernfunktionen wie z.B. Quickprop oder Backprop_Momentum führte zu keiner Verbesserung des gelernten Netzes.

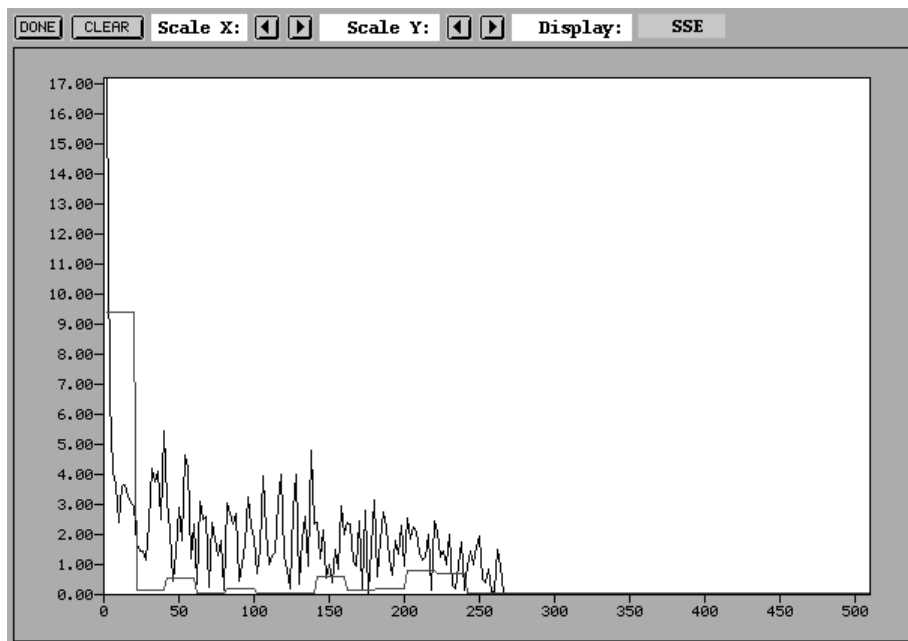


Abb.3 Lerngraph eines Netzes mit Act_Logistic Aktivierungsfunktion jedoch mit Quickprop Lernfunktion

4. Programmtechnische Struktur

4.1. Modularer Aufbau

Das fertige Programm besteht aus drei Modulen. Die auf Qt basierende Oberfläche realisiert sämtliche Interaktionen des Nutzers mit dem Programm, die speziell entwickelte "lib_pics"-Bibliothek in der alle Routinen zur Verarbeitung und Erkennung des Logos enthalten sind, sowie die Schnittstelle für das im SNNS trainierte Netz.

Mit Hilfe des vom SNNS mitgelieferten Tools "snns2c" wurde eine C-Bibliothek aus einem angegebenen neuronalen Netz erzeugt, die es ermöglicht, für eine in das trainierte Netz gegebene Inputmatrix, den berechneten Outputwert zu ermitteln.

4.2. Die "lib_pics" Bibliothek

Die Bibliothek erfüllt mehrere Aufgaben. Sie lädt die JPG Bilder in den Speicher und wandelt sie dort in eine float Matrix um. Dabei werden die Farbinformationen der einzelnen Pixel des Bildes nach 16 Farben normiert und diese auf den Inputwertebereich im Intervall [0;1] skaliert.

Anschließend wird mit Hilfe verschiedener Verfahren in der Matrix das Logo gesucht. Die Bibliothek stützt sich dabei auf das von uns entwickelte Scanline Verfahren, dem Fullscan sowie dem etwas genaueren Simplescan. Die beiden letzteren Scanverfahren greifen dabei auf das vom SNNS erzeugte und in eine C-Bibliothek umgewandelte Netz zurück.

4.2.1. Das Scanline Verfahren

Mit Hilfe des Scanline Verfahrens ist es möglich, unabhängig vom trainierten Netz die relative Position des Logos zu bestimmen. Das Verfahren beruht dabei auf der Annahme, daß in vielen Fällen das Logo eine andere Farbe hat als das umgebende Fernsehbild.

Deshalb wird als erstes eine Normierung des Bildes entsprechend der Pixelfarben durchgeführt. Jedes Pixel das nicht der Farbe des Logos entspricht wird nach Null normiert, wogegen jedes Pixel, welches die Logo-Farbe beinhaltet, auf eins gesetzt wird. Anschließend wird das Verhältnis zwischen den beiden Pixelgruppen errechnet. Entspricht dieses Verhältnis nicht annähernd dem eines idealen Bildes³, bricht das Verfahren ab.

Durch diese Umwandlung ist gewährleistet, daß sich nur noch das Logo selbst und eventuell einzelne "versprengte" Pixel im Bild befinden. Es wird damit

³ein Bild, daß nur aus schwarzen Pixeln für die Umgebung und dem Logo selbst besteht

aber auch der Nachteil des Verfahren offensichtlich, denn es eignet sich nur für Bilder mit ein- bzw. mehrfarbigen Logos, nicht jedoch für Bilder mit halbdurchsichtigen oder animierten Logos.

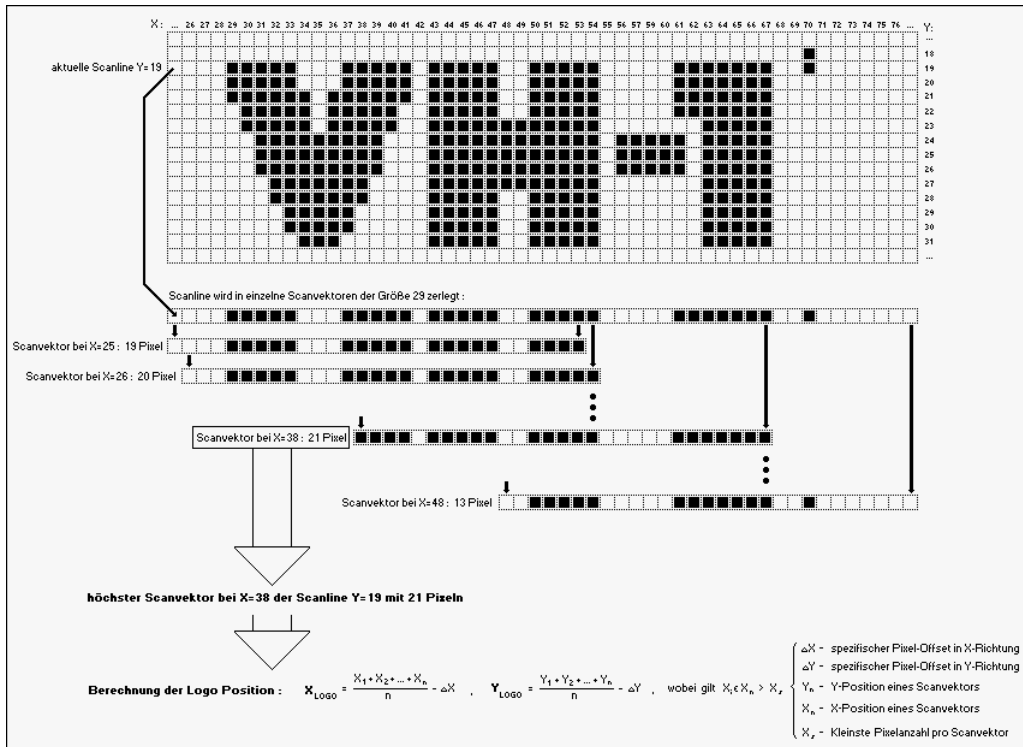


Abb.4 Struktur des Scanline Algorithmus

Nachdem das Bild erfolgreich normiert wurde, startet jetzt das eigentliche Scan Verfahren (siehe Abb. 4). Jede Zeile (sogenannte Scanline) des Bildes wird in sich überlappende Pixel-Vektoren (sogenannte Scanvektoren) einer bestimmten Größe zerlegt (bei VH1 Logos z.B. 29 Pixel). Die Größe der Vektoren hängt von der Größe des jeweiligen Logos des TV-Senders ab. Anschließend wird der Vektor mit der höchsten Anzahl an Pixeln des Wertes eins ermittelt. Wenn der Anteil dieser Pixel einen bestimmten Schwellwert überschreitet, wird sich die X-Position des Vektors in der Scanline gemerkt. Sind alle Zeilen des Bildes gescannt worden, wird die wahrscheinliche Position des Logos nach folgedner Formel berechnet:

$$X_{logo} = \frac{X_1 + X_2 + \dots + X_n}{n} - \Delta X$$

$$Y_{logo} = \frac{Y_1 + Y_2 + \dots + Y_n}{n} - \Delta Y$$

$$X_i \in X_n > X_s$$

ΔX und ΔY sind dabei Offsetwerte um die linke obere Ecke des Logos zu bestimmen. Sie hängen von der jeweiligen Größe des Logos eines TV-Senders ab.

4.2.2. Das Fullscan Verfahren

Dieses Verfahren ist das eigentliche Scan Verfahren unter Zuhilfnahme des neuronalen Netzes. Dabei wird in einer Schleife jede mögliche Position der Inputmatrix im Ausgangsbild dem trainierten neuronalen Netz übergeben. Wenn der Outputwert des Netzes einen bestimmten Schwellwert überschreitet, wird die x und y Position der Inputmatrix bezüglich des Ausgangsbildes gespeichert. Anschließend wird ein Mittelwert sämtlicher gefundener Positionen gebildet.

4.2.3. Das Simplescan Verfahren

Das Simplescan Verfahren dient zur Präzisierung der Ergebnisse aus dem Fullscan oder dem Scanline Verfahren. Im Radius von fünf Pixeln zur dort ermittelten Position wird erneut die Inputmatrix in das neuronale Netz gegeben. Der höchste Outputwert des Netzes ist gleichzeitig die endgültige Position des Logos im Bild.

4.2.4. Die Berechnung im Detail

Die Kombination aller drei Verfahren, niedergelgt in der `run_calc()` Funktion (siehe Abb. 5), ermöglicht die optimale Erkennung des Logos im Bild. Zuerst kommt das Scanline Verfahren zum Einsatz. Falls dieses keine Position des Logos ermitteln kann, wird mit Hilfe des Fullscan Verfahrens erneut versucht, eine Logoposition zu ermitteln. Versagt auch dieses Verfahren, wird zunächst angenommen, daß ein Logo im Bild vorhanden ist, es jedoch nicht ermittelt werden konnte. Kann jedoch nach einer bestimmten Anzahl von Versuchen an unterschiedlichen Bildern (Standardwert ist 2) weiterhin kein Logo ermittelt werden, schlußfolgert die Bibliothek, daß kein Logo in den bisher negativ geprüften Bildern enthalten ist. Es handelt sich aller Wahrscheinlichkeit nach um eine Werbesendung.

Ist dagegen das Fullscan oder das Scanline Verfahren erfolgreich, wird die gefundene Position an das Simplescan Verfahren übergeben. Dieses ermittelt dann die endgültige Position des Logos. Die Bibliothek schlußfolgert dann, daß das normale TV-Programm läuft.

Wird dieser Zustand einmal erreicht, merkt sich die Bibliothek die Position

des Logos. Bei jedem neuen Bild wird nun nur noch das Simplescan Verfahren aufgerufen. Erst wenn es ein Logo nicht findet, wird die gespeicherte Logoposition gelöscht und die `run_calc()` Funktion beginnt von vorn.

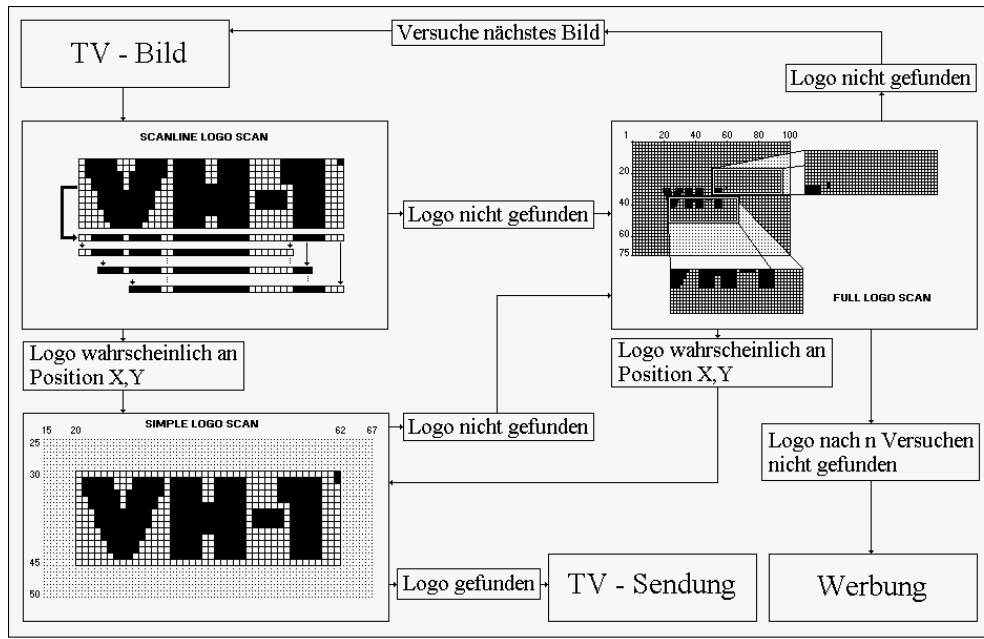


Abb.5 Struktur der Run_Calculate Funktion

4.3. Die Oberfläche

Die Oberfläche wurde mit Hilfe der Qt-Bibliothek der Firma Troll Tech erstellt. Das Layout wurde über das Tool "qtarch" (der Qt-Architekt) realisiert. Hinter sämtlichen Buttons und Textfeldern befindet sich ein virtueller Slot, der mit verschiedenen Funktionalitäten in einer abgeleiteten Klasse gefüllt wurde. Dabei sind alle Funktionen direkt mit der "lib_pics" Klasse verbunden. Die Qt-Bibliothek ist in der Programmiersprache C++ geschrieben und ließ sich relativ einfach mit den bestehenden Bibliotheken verbinden. Leider greift Qt wie unsere selbgeschriebene "lib_pics" Bibliothek auf die JPEG Klasse der Independent JPEG Group zurück. Deshalb waren einige Modifikationen in den Header Dateien der JPEG-Klasse nötig. Darüber hinaus wandelt Qt beim Laden von JPG Dateien diese in ein proprietäres Grafikformat im Speicher um. Es ist somit nicht möglich, exakte Farbinformationen einzelner Pixel bezüglich einer Standardfarbpalette aus diesen Grafikdaten zu erfahren. So blieb uns nur die Möglichkeit, JPG Bilder in unserer "lib_pics" Bibliothek gesondert zu laden und dort weiter zu verarbeiten.

5. Installation und Bedienung

5.1. Systemanforderungen

Das Programm benötigt ein Linux mit einer Kernelversion ab 2.0, 64 MByte RAM und 20 MByte freien Festplatterspeicher. Es müssen weiterhin folgende Pakete installiert sein:

- ein funktionierender X-Server mit einer Auflösung von 800x600 Pixeln bei 16 Bit Farbtiefe
- ein C++ Compiler (empfehlenswert "g++" ab Version egcs-2.91.66) inklusive Linker
- Qt Library ab der Version 1.41
- JPEG Library der Independent JPEG Group Version 6b
- Archivierungsprogramm tar und der Entpacker gunzip
- ImageMagick Version 5.1.0-36

Die Entwicklungs- und Testumgebung bestand aus einem Pentium II 400 MHz mit 128 MByte RAM (S.u.S.E. Linux 6.0), einem AMD K6/2 350 MHz (S.u.S.E. Linux 6.2) mit 64 MByte RAM und einem AMD Thunderbird 800 MHz (S.u.S.E. Linux 6.4) ebenfalls mit 128 MByte RAM.

Auf allen 3 Systemen waren neben den oben genannten Paketen noch die Pakete SNNS Version 4.1 und der Qt Architect Version 1.4 installiert. Der Qt Architect benötigt außerdem das Paket "tmake" ab der Version 1.2. Diese Pakete sind nur für die weitere Entwicklung der Software von Bedeutung.

Aus Performance Gründen ist ein Prozessor neuerer Generation mit mindestens 400 MHz empfehlenswert.

5.2. Installation der Software

Das Programmarchiv `tvlogo.tar.gz` installiert sich durch folgende Kommandos:

- `gunzip tvlogo.tar.gz`
- `tar xvf tvlogo.tar`

Aus dem tar Archiv `tvlogo.tar.gz` ergibt sich folgende Verzeichnisstruktur (siehe Abb.6). Das Verzeichnis `tvlogo` wird im aktuellen Verzeichnis angelegt und ist das Wurzelverzeichnis der Anwendung. Hier befindet sich ein Script

namens `tvlogo`, welche die ausführbare Datei `bin/tvlogo` aufruft. Desweiteren befindet sich diese Dokumentation als Dateien im `.dvi`⁴ und im `.pdf` Format.

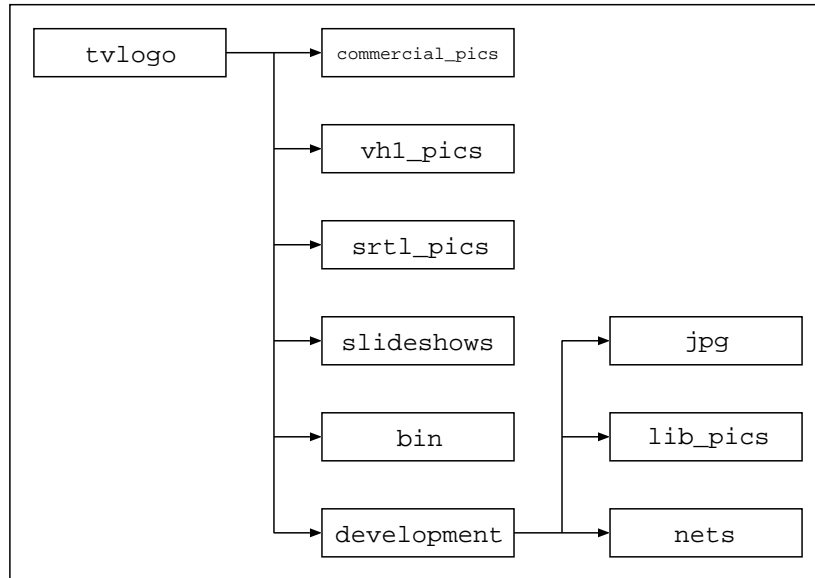


Abb.6 Struktur des Verzeichnisbaumes

In den Verzeichnissen `commercial_pics`, `srtl_pics` und `vh1_pics` ist eine große Anzahl von JPG Bildern abgelegt, die als Beispiele für das Programm dienen. Sie sind in die jeweiligen Verzeichnisse als Werbebilder oder Bilder eines Fernsehsenders sortiert. Beispiele für die Folge mehrerer Bilder, sogenannte Slideshows, findet man dagegen im Verzeichnis `slideshows`.

Das Verzeichnis `bin` beinhaltet die eigentlichen Programmdateien. Hier befindet sich das `makefile`, welches eine ausführbare Programmdatei erstellt, wenn in der Kommandozeile einfach der Befehl `make` eingegeben wird. Die startbare Datei trägt den Namen `tvlogo`.

Im letzten Verzeichnis `development` sind alle Dateien gespeichert, die zur weiteren Entwicklung des Programms nötig sind. Die Dateien zur Erweiterung der JPEG Library befinden sich im Unterverzeichnis `development/jpg`. Sie stellen einige wichtige Funktionen zum Umgang mit JPG Bildern bereit und wird deshalb von der "lib_pics" Bibliothek eingebunden. Die Bibliothek befindet sich im Verzeichnis `bin`. Die Entwicklungsversion der Bibliothek findet man dagegen im Verzeichnis `development/lib_pics`.

Weiterhin sind alle trainierten und untrainierten Netze sowie die zugehörigen Pattern Dateien im Verzeichnis `development/nets` abgelegt.

⁴TeX Bitmap output Format - z.B. betrachtbar mit `xdvi`

5.3. Bedienung der Oberfläche

Die Bedienung des Programmes ist recht einfach und intuitiv. Die Oberfläche besteht aus 4 Hauptkonsolen mit unterschiedlichen Aufgaben. Die TV-Konsole auf der linken Seite bezieht sich auf alle TV-spezifischen Einstellungen und dem Anzeigen des Programmes. In der Mitte ist die Output-Konsole, die alle Programminformationen für den Anwender bereitstellt. Unter diesen beiden Konsolen befindet sich die Pattern-Konsole zum Erstellen von Pattern Dateien. Auf der rechten Seite findet man die Bedien-Konsole mit der fast alle Aktionen des Programmes gesteuert werden können.

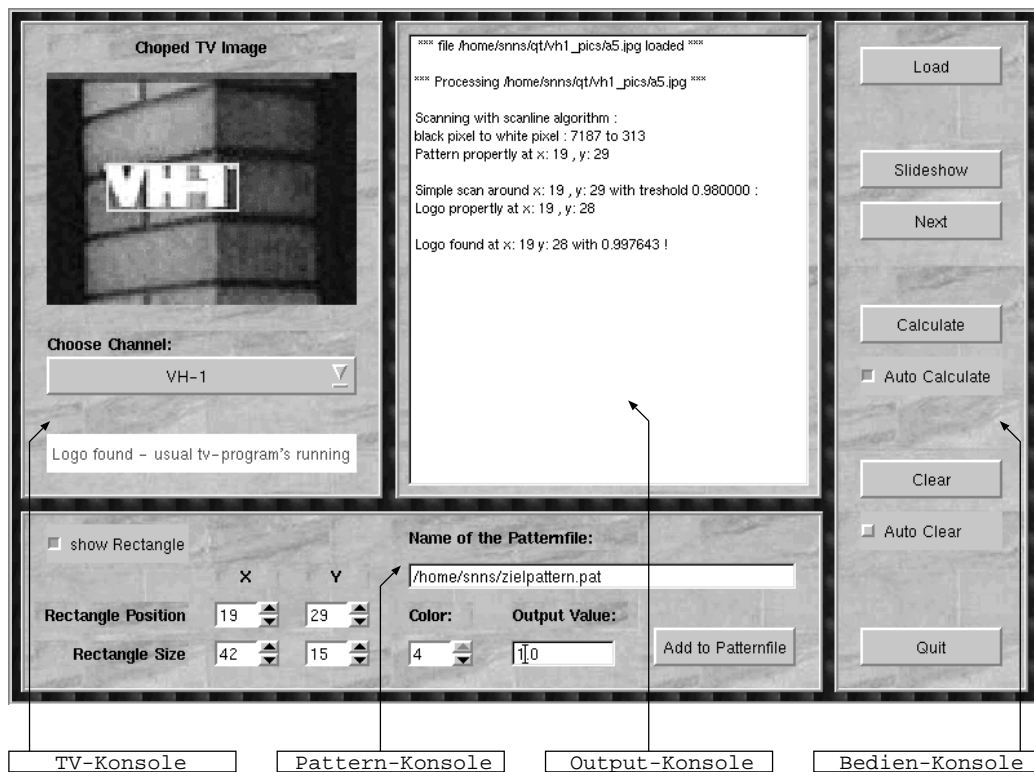


Abb.7 Oberfläche des Programmes

5.3.1. Die TV- und die Output-Konsole

Die Anzeige des aktuellen Bildes, eine Listbox zur Auswahl des aktuellen Kanals sowie die Statusanzeige des Programmes beinhaltet die TV-Konsole. Über die Auswahl des TV-Kanals ist festgelegt mit welchem neuronalen Netz das Programm das aktuelle Bild analysiert. Jeder TV-Kanal hat dabei sein eigens, speziell angepasstes neuronales Netz. Die Statusanzeige informiert darüber, ob auf dem geladenen Bild ein Logo erkannt wurde und somit das

normale TV-Programm läuft oder es sich um Werbung handelt. Weiterhin gibt sie durch kurze Statusmeldungen an, in welchem Zustand sich das Programm im Moment befindet.

Die Output-Konsole zeigt dagegen alle Zwischenmeldungen des Programmes an. Sie gibt dabei auch die genaue Position des gefundenen Logos aus.

5.3.2. Die Bedien-Konsole

Mit Hilfe der Bedien-Konsole ist es möglich wesentliche Funktionen des Programmes zu steuern. Auf dieser Konsole befinden sich verschiedene Buttons und Schalter, die im folgenden erklärt werden:

- Der **Load**-Button öffnet einen File Dialog mit dessen Hilfe eine JPG Datei, welche das TV-Bild repräsentiert, in das Programm geladen wird.
- Der **Slideshow**-Button öffnet einen File Dialog zum Laden einer Slideshow. Eine Slideshow ist dabei eine Anzahl von Bilddateien mit dem gleichen Namen, aber jeweils unterschiedlichen Endungen. Sie beginnt immer mit Bild *slideshow.0* und lädt das Folgebild mit der um eins erhöhten Endung nach (*slideshow.1*, *slideshow.2*, ...). Dabei sind die einzelnen Bilder nichts weiter als entsprechend umbenannte JPG Dateien. Durch diese Funktion soll der Ablauf eines echten TV-Programmes simuliert werden und die Funktionalität des Programmes demonstriert werden.
- Der **Next**-Button lädt das Folgebild einer Slideshow, vorausgesetzt, daß eine solche gestartet wurde. Findet die Slideshow kein Folgebild mehr, beginnt sie von vorn.
- Der **Calculate**-Button startet die Analyse des aktuellen Bildes.
- Der **Auto Calculate**-Schalter veranlaßt das Programm sofort nach dem Laden eines Bildes die Analyse zu starten. Dieser Schalter wird automatisch aktiviert, falls eine Slideshow gestartet wird.
- Der **Clear**-Button löscht den Inhalt der Output-Konsole.
- Der **Auto Clear**-Schalter veranlaßt das Programm die Output-Konsole vor jeder neuen Analyse zu löschen.
- Über den **Quit**-Buttons wird das Programm beendet.

5.3.3. Die Pattern-Konsole

Die Pattern-Konsole dient zum eigenen Erstellen von Pattern Dateien. Sie ist somit als Unterstützung bei der Entwicklung neuer Netze gedacht. Mit Hilfe des Schalters **Show Rectangle** läßt sich ein Rechteck einblenden, das über die darunter liegenden Bedienelemente "Rectangle Position X,Y" bzw. "Rectangle Size X,Y" mit beliebiger Größe im Bild positioniert werden kann. Zur besseren Sichtbarkeit des Rechteckes, läßt sich dessen Farbe unter "Color" ändern. Es stehen dabei vier Grundfarben zur Verfügung.

Das Rechteck stellt den Bereich der Input-Schicht dar, die dem Netz gelernt werden soll. Durch Klick auf den Button **Add to Patternfile** wird der im Rechteck ausgewählte Bereich der unter "Name des Patternfiles" spezifizierten Pattern Datei hinzugefügt. Vorher sollte jedoch der zu lernende Outputwert der Netzes im Feld "Output Value" eingegeben werden.

Während des Speicherns von Pattern in eine Dateien ist besonders darauf zu achten, daß die Werte für die Rechteckgröße nicht geändert werden. Ansonsten werden Pattern mit unterschiedliche Größen in eine Pattern Datei gespeichert, was diese unbrauchbar macht. Dasselbe gilt beim Anhängen von Patterns an eine bereits bestehende Datei. Die richtige Rechteckgröße wird vom Programm nicht automatisch erkannt und sollte deshalb korrekt eingestellt sein, bevor mit dem Anhängen von Daten begonnen wird.

Die fertige Patterndatei kann nun wiederum in die **SNNS** Software geladen werden und dem Netz trainiert werden. Ist das Netz fertig trainiert, kann es mit Hilfe des **snns2c**-Tool wieder in eine C-Bibliothek umgewandelt werden. Diese Bibliothek kopiert man in das **bin** Unterverzeichnis und kompiliert das Programm mit dem Befehl **make** erneut. Voraussetzung dafür ist jedoch, daß das neue Netz den selben Namen hat wie das bisherige.

6. Bewertung der erreichten Lösung

6.1. Leistungsfähigkeit

Die erreichte Lösung erfüllt die Zielstellung vollständig. Das Programm ist in der Lage, in Bildern Fernsehlogos mit hoher Wahrscheinlichkeit zu erkennen. Bei Testläufen mit über 300 VH-1Bildern wurden lediglich 15 Bilder falsch erkannt. Bei genauerer Betrachtung dieser Bilder war es jedoch selbst dem menschlichem Auge nicht immer möglich, das Logo einwandfrei zu erkennen. Meist war es so stark deformiert oder verrauscht, daß es ganz verschwand. Ein ähnliches Ergebnis erreichten wir auch bei einem Test mit 300 Super RTL-Bildern (12 Fehlerkennungen). Die Leistungsfähigkeit der trainierten Netze

wird besonders dadurch unterstrichen, daß bei der Analyse von 300 Bildern ohne Logo auf keinem Bild ein Logo fälschlicherweise erkannt wurde. Die Netze erkennen also ca. 97% der ihnen gezeigten Bilder korrekt. Denkt man sich noch die Toleranz des Programmes gegenüber einmaliger Fehlerkennungen der Netze hinzu, fällt das Programm zu 99% die korrekte Entscheidung ob gerade eine Werbesendung oder das normale Fernsehprogramm läuft.

6.2. Erweiterungen

Obwohl das Programm die Aufgabenstellung erfüllt, gibt es noch zahlreiche Möglichkeiten der Erweiterung. Größter Nachteil der derzeitigen Version der Software ist wohl, daß sie nur die Logos zweier TV-Sender erkennt. Es müßten also weitere TV-Sender hinzugefügt werden.

Daneben kann, wie weiter oben schon beschrieben, die Netztopologie verändert werden, um eine höhere Performance des Netzes zu erreichen. Auch der Aufbau der Patternfiles kann verändert werden. So könnte mit Zwischenstufen eines Logos erreicht werden, daß das Netz besser assoziieren kann. Als besonders sinnvoll und leistungssteigernd ist sicherlich auch der Einsatz eines zweiten Netzes pro TV-Kanal, daß ausschließlich zur Vorerkennung der relativen Logo-Position gedacht ist. Das langsame Fullscan-Verfahren würde somit seltener zum Einsatz kommen.

Eine zusätzliche Erweiterung wäre das Erkennen von transparenten Logos. Hierzu müßte zur Vorverarbeitung der Bilder ein grafischer Filter zum Einsatz kommen, der die Strukturkanten eines Bildes hervorhebt. Der sogenannte "emboss"-Filter des Imagemagick Paketes, welches das Gradientenverfahren implementiert, erfüllt diese Forderung. Die umgewandelten Bilder sind meist jedoch ein relatives Chaos aus verschiedenen Kantenübergängen und müssen deshalb mit einem speziell dafür trainierten Netz vorverarbeitet werden. Auch ein neues Scanline-Verfahren würden benötigt. Es muß dabei nicht mehr nach zusammenhängenden Vektoren, sondern nach besonderen, auf das Logo angepasste, Strukturen suchen.

Neben den Erweiterungen der Analyse von Bildern, könnte auch die Funktionalität des Programmes dahingehend geändert werden, daß es auf den unter Linux zur Verfügung stehend TV-Karten-Treiber **bttv** zurückgreift. Einer Echtzeitanalyse des TV-Programmes stände dann nichts mehr im Weg. Würde dann noch ein Videorekorder per serieller Schnittstelle bzw. ein MPEG3-Videograbber angesteuert, wäre der erste intelligente werbungausblendende Videorekorder geboren.

Bei dem derzeitigen Stand der Software und vor allem der Leistungsfähigkeit heutige Rechner ist das wohl eher eine Illusion.

Anhang A

A.1. Dateiliste

Dateien im Verzeichnis `bin` :

Dateiname	Beschreibung
TvQtApp.cpp TvQtApp.h	Programm und Headerdatei zum realisieren sämtlicher Funktionen der Oberfläche, abgeleitet von TvQtAppData
TvQtAppData.cpp TvQtAppData.h	Programm und Headerfile, welche sämtliche Layouteinstellungen beinhalten
back1.bmp back3.bmp	Bitmap Dateien zum texturieren der Bedienoberfläche
jmorecfg.h	Beinhaltet Konfigurationsdefintionen für die JPEG Bibliothek. Aufgrund einiger Konflikte mit Qt mußte diese Datei modifiziert werden. (c) 1991-1996 Thomas G.Lane
jpeglib.h	Beinhaltet Definitionen für die Applikationsschnittstelle der JPEG Bibliothek. Aufgrund einiger paralleler Typedefinitionen mit Qt ist diese modifizierte Headerdatei hier nochmals abgelegt (standardmäßig unter <code>/usr/lib</code>). (c) 1991-1996 Thomas G.Lane
lib_pics.cp lib_pics.h	Programm und Headerdatei der "lib_pics"-Bibliothek zur Analyse der TV-Bilder und Abspeichern von Pattern Dateien.
makefile	Dient zum Kompilieren des tvlogo Programmes. Es stützt sich dabei auf die Datei makefile.pro . Das makefile wurde dabei mit dem Qt-Tool tmake angelegt und im weiteren Verlauf von Hand modifiziert.
makefile.pro	Vom Qt-Architekten für tmake angelegtes Template-Makefile. Es wird unbedingt für die "moc"-Operation der Qt Bibliothek im makefile benötigt.

qjpegio.cpp qjpegio.h	Programm und Headerdatei des QImage IO-Handler der Qt Bibliothek. Diese Datei wird benötigt um JPG Dateien in das Qt proprietäres Grafikformat während des Ladens umzuwandeln.
showimg.cpp showimg.h	Programm und Headerdatei zum Anzeigen eines QImage-Bildes. Weiterhin realisieren sie das Zeichnen und Anzeigen des Pattern-Rechtecks im QImage-Bild.
srtl_super_weighted2.c srtl_super_weighted2.h	Trainiertes und mit snns2c in eine C-Bibliothek umgewandeltes neuronales Netz zum Erkennen von Super RTL-Logos.
tvlogo	Fertig kompiliertes Binary - also das Programm an sich.
tvlogo.cpp	Hauptprogramm, welches sämtliche im Verzeichnis <code>bin</code> liegende Bibliotheken und Unterprogramme in sich vereint. Weiterhin greift es auf die JPEG Bibliothek im Verzeichnis <code>/development/jpg</code> zurück.
tvlogo.dlg tvlogo.prj	Dialog und Projekt Datei der Oberfläche für den Qt-Architekten qtarch . Wird benötigt falls man Veränderungen an der Oberfläche im Architekten vornehmen will.
vh1_1_weighted.c vh1_1_weighted.h	Trainiertes und mit snns2c in eine C-Bibliothek umgewandeltes neuronales Netz zum Erkennen von VH1-Logos.

Dateien im Verzeichnis `development/lib_pics` :

Dateiname	Beschreibung
<code>conv</code>	Shell Skript zum Umwandeln eines, zuvor in einer BMP Datei abgespeicherten, vollständigen Fernsehbildes in eine JPG Datei. Dabei wird nur die entsprechende Ecke des Logos in einer Matrix von 200x100 Pixel ausgeschnitten und anschließend auf 100x75 Pixel skaliert. Das Skript stützt sich dabei auf die Programme convert und mogrify des ImageMagick Programmpaketes. Bei den BMP Dateien handelt es sich um von der TV-Karte abgespeicherte Windows-Bitmap-Dateien.
<code>conv_all</code>	Shell Skript zum knovertieren sämtlicher BMP Fernsehbild-Dateien in einem Verzeichnis in JPG-Dateien. Das Skript ruft dabei jedesmal das conv -Skript auf.
<code>lib_main</code>	Das "lib_pics"-Bibliothek Testprogramm als fertig kompiliertes Binary.
<code>lib_main.cpp</code>	Hauptprogramm, zum Testen der Funktionalität der "lib_pics"-Bibliothek. Es greift außerdem auf die JPEG Bibliothek im Verzeichnis <code>/development/jpg</code> zurück.
<code>lib_pics.cpp</code> <code>lib_pics.h</code>	Programm und Headerdatei der Entwicklungsversion der "lib_pics"-Bibliothek. Sie dient zur Analyse der TV-Bilder und dem Abspeichern von Pattern Dateien.
<code>makefile</code>	Dient zum Kompilieren des lib_main Programmes.
<code>srtl_super_weighted2.c</code> <code>srtl_super_weighted2.h</code>	Trainiertes und mit snns2c in eine C-Bibliothek umgewandeltes neuronales Netz zum Erkennen von Super RTL-Logos.

vh1_1_weighted.c vh1_1_weighted.h	Trainiertes und mit snns2c in eine C-Bibliothek umgewandeltes neuronales Netz zum Erkennen von VH1-Logos.
--------------------------------------	--

Dateien im Verzeichnis `development/jpg` :

Dateiname	Beschreibung
graf.h	Beinhaltet alle Struct Definitionen zum Laden und Umwandeln von JPG Bildern. (c) 1991-1996 Thomas G.Lane
jinclude.h	Behebt Fehler die beim Zusammenspiel mit der JPEG Bibliothek auftreten. (c) 1991-1996 Thomas G.Lane
x_jdatasrc.c x_jdatasrc.h	Programm und Headerdatei zum Verwalten von JPG Streams. (c) 1998 G.Brand
x_read_jpeg.c x_read_jpeg.h	Programm und Headerdatei zum Einlesen der JPG Daten. (c) 1998 G.Brand
x_src_class.h	Beinhaltet Struct Definition der JPG Source Streams. (c) 1991-1996 Thomas G.Lane
x_src_file.c x_src_file.h	Programm und Headerdatei für JPG Basis Datei-Routinen. (c) 1998 G.Brand
x_src_stream.c x_src_stream.h	Programm und Headerdatei für erweiterte JPG Datei-Routinen. (c) 1998 G.Brand