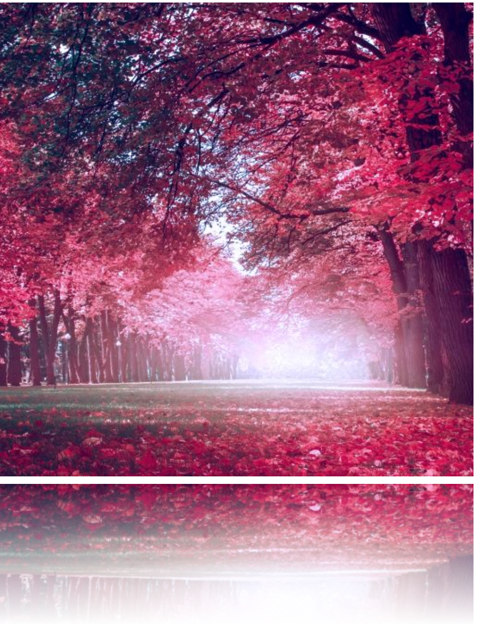
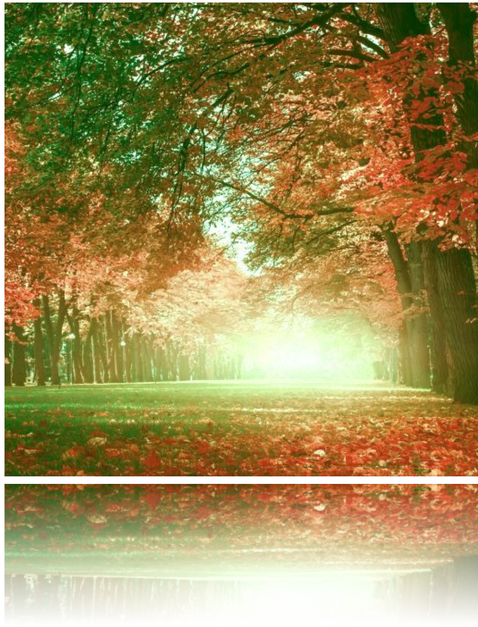


PICO

Prolog Image Customization Optimizer



High Level languages: Prolog WS1718

Our Idea

Fun things to do with Linear Algebra and Prolog?

- Lets do Image Processing!
  - Feature Extraction
  - Image Filters

Make use of Prolog

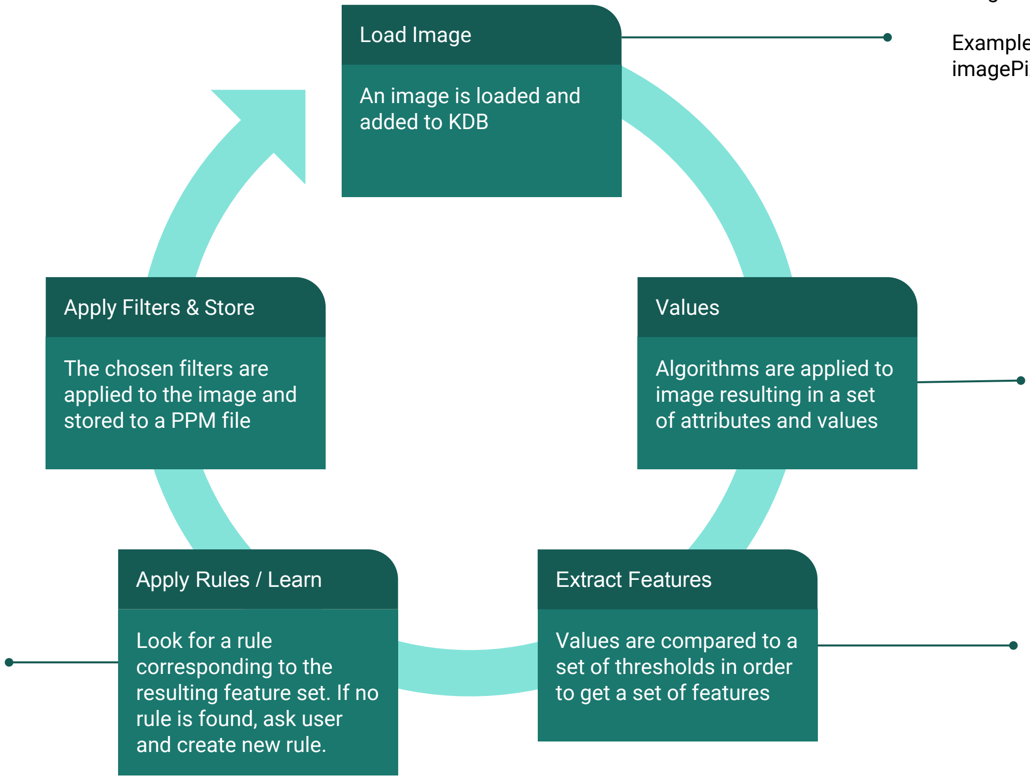
Make it smart!

- Rules that decide which filters are applied
- Learning? Two Programs? Training and Processing?

A whole folder is iterated through and with each new image the program learns from the user’s opinion.

Demo

Image Processing Loop



Load Image

An image is loaded and added to KDB

Values

Algorithms are applied to image resulting in a set of attributes and values

Extract Features

Values are compared to a set of thresholds in order to get a set of features

Apply Rules / Learn

Look for a rule corresponding to the resulting feature set. If no rule is found, ask user and create new rule.

Apply Filters & Store

The chosen filters are applied to the image and stored to a PPM file

imagePixel(Id,H,W,R,G,B)

Example :
imagePixel(('4.ppm', 1,1,9,9,9)

imageValue(id, attribute1, value1)
imageValue(id, attribute2, value2)

Example :
imageValue('4.ppm',
maxBrightness, 300).
imageValue('4.ppm', stdR, 4000).

feature(id, feature1)

Example :
feature('4.ppm', colorful).

filter(id, filter1)

Example :
filter('4.ppm', blur).

PICO Main Loop (1)

```
:- set_prolog_flag(verbose, silent).
:- op(1099, xfx, --->).
:- op(1199, xfx, ::).
:- op(999, fx, *>).
:- op(999, fx, ~>).
:- op(999, fx, *<).
:- op(999, fx, ~<).
:- op(999, fx, *>=).
:- op(999, fx, *=<).
:- op(999, fx, *+).
:- op(998, xfx, *+).
:- dynamic((--->)/2).
:- dynamic((::)/2).
```

```
:- dynamic(debug_mode/0).
:- dynamic(current_id/1).
:- use_module(library(lists)).
:- use_module(library(filesex)).
:- ensure_loaded(util).
:- ensure_loaded(fileUtil).
:- ensure_loaded(imageUtil).
:- ensure_loaded(imageFilters).
:- ensure_loaded(imageLoader).
:- ensure_loaded(imageWriter).
:- ensure_loaded(thresholds).
:- ensure_loaded(executor).
:- initialization main.
```

PICO Main Loop (2)

```
% main entrance into the application
main :-
    !,log('--Program-Start--'), nl,
    !,listPPMInputFiles(ImageIdList),
    !,process(ImageIdList), nl,
    !,info('finished').

% application error case
main :- info("error").
```

PICO Main Loop (3)

```
% Main Image Processing Loop
process([]):- info("Gone through all images").
process([ImageId|ImageIdList]) :-
    % Loading the image into Prologs DB
    !,info('####################'),
    !,info('Loading Image'),
    !,string_concat('./img/input/',ImageId,FilePath),
    !,loadImage(FilePath,ImageId),
    !,info('Describing Image:'),
    !,describeImage(ImageId),
    !,info('Finished Describing Image'),
    !,info('Relevant Values: '),
    % Extract the feature list from the value list
    % created by finding all imagevalues in the DB
    !,findall(imageValue(ImageId,Key,Value),
      imageValue(ImageId,Key,Value), ValueList
    ),
    !,extractFeatures(ValueList),
```

```
% Check if a Rule already exists
!,(checkRules(ImageId, Filter) ->
    (sinfo('Found Exisiting Rule. Applying'),
     info(Filter)
    );
    (makeNewRule(ImageId, Filter))
), nl, nl,
info('Current Rule Set:'),
listing(--->),
% apply the found filter to current Image
applyFilter(Filter,ImageId,ResultImageId),
writeOutput(ResultImageId),
deleteImage(ImageId),
deleteImage(ResultImageId),
!,process(ImageIdList).
```

Important Util - Custom Logger

```
debugEnabled(global).
%debugEnabled(util).
%debugEnabled(imageWriter).
%debugEnabled(imageLoader).
%debugEnabled(imageValues).
%debugEnabled(sobelEdge).
%debugEnabled(imageUtil).

% advanced logging feature that can be disabled with a global fact
debug(CodeIdentifier,Reason) :-
   % only log if it is enabled in config
   debugEnabled(CodeIdentifier),
   ansi_format([fg(yellow)], '~w |> ~w', [CodeIdentifier,Reason]), nl.
debug(_,_). % ignore it
```

Important Util - PPM Text Parser

```
% reads a word from a stream / stops at blank and eof - ignores newlines
readWordIgnoreNewLine(InStream,W):-
    get_code(InStream,Char),
    readWordIgnoreNewLineFilter(Char,Chars,InStream),
    atom_codes(W,Chars).


% filter used for readWordIgnoreNewLine ignore chars
readWordIgnoreNewLineFilter(32,[],_):-  !. % blank
readWordIgnoreNewLineFilter(-1,[],_):-  !. % eof
readWordIgnoreNewLineFilter(end_of_file,[],_):-  !.
readWordIgnoreNewLineFilter(10,Chars,InStream):- % prevents newlines stopping parsing
    get_code(InStream,NextChar), readWordIgnoreNewLineFilter(NextChar,Chars,InStream).
readWordIgnoreNewLineFilter(Char,[Char|Chars],InStream):-
    get_code(InStream,NextChar), readWordIgnoreNewLineFilter(NextChar,Chars,InStream).
```

Important Util - PPM File Finder

```
% list all ppm files in the input folder
listPPMInputFiles(InputFiles):- directory_files("./img/input",RawInputFiles),
                                 filterPPMs(RawInputFiles,InputFiles,[]).

% filter out valid .ppm files from a folder
filterPPMs([],   Output,Acc):- Output = Acc.
filterPPMs([H|T],Output,Acc):- name_value(H,_,"ppm"),
                               append(Acc,[H],NewAcc),
                               filterPPMs(T,Output,NewAcc).
filterPPMs([_|T],Output,Acc):- (T,Output,Acc).

% used to split string into a name and value field, devided by a "."
% this is used to search for all .pmm files in the relevant folder
name_value(String, Name, Value) :-
        sub_string(String, Before, _, After, "."), !,
        sub_string(String, 0, Before, _, Name),
        sub_string(String, _, After, 0, Value).
```

Loading / Writing the PPM Images

```
P3
# Created by IrfanView
8 8
255
60 47 29 68 55 37 90 59 48 109 55 43 108 47 38
100 38 33 77 26 25 72 22 21 76 48 30 54 47 29
79 51 37 101 49 39 122 53 42 144 55 45 155 39 37
90 16 18 111 54 35 88 56 38 81 54 38 124 83 73
161 88 78 162 70 54 168 51 46 108 18 20 129 46 34
124 50 34 126 58 43 159 121 109 166 94 76 163 77 57
181 64 52 154 37 37 171 60 52 194 90 76 192 91 74
225 149 140 195 138 121 182 99 76 161 73 59 91 31 26
134 65 49 158 95 69 209 145 128 234 179 170 239 205 199
229 192 179 199 133 114 102 45 34 111 79 48 141 104 66
167 131 91 198 163 131 223 196 176 226 195 176 201 147 119
120 58 37 82 33 18 102 41 24 123 46 29 145 53 36
174 65 49 196 72 61 193 62 53 184 47 43
```

- PPM Image Format
  - Simple Image Format
  - Supports ANSI Content
- Keeps the Image Parsing simple
  - RRR GGG BBB RRR GGG BBB…
- Problem with writing parser
  - Using FileStream
  - Performance -> DiffLists needed
  - Undefined parsed images

High Performance Image Loader (1)

```
% Loads a Image file at path and returns the Image in a custom format
% open and closing the file is independent of the processed result
loadImage(Path,Image):-
    setup_call_cleanup(
        (
         debug(imageLoader,'Opening File'),
         open(Path, read, FileStream,[buffer(full),close_on_abort(true)])
         ),
        (!,processLoadImageStream(FileStream,Path, Image),!),
        (debug(imageLoader,'Closing File'),close(FileStream))
    ).
```

High Performance Image Loader (2)

```
% loads a filestream into an image and stores it into
% 1) imageMeta(ImageId,width,height,depth) 2) imagePixels(ImageId,x,y,r,g,b)
processLoadImageStream(FileStream,Path,ImageId) :-
    debug(imageLoader,'Loading Header'),
    readWord(FileStream,FileBang),
    FileBang == 'P3', % check filebang
    debug(imageLoader,'Loading Comment Field'),
    readLine(FileStream,_),
    debug(imageLoader,'Loading Image Spec'),
    readWord(FileStream,WidthStr),  atom_number(WidthStr,  Width),
    readWord(FileStream,HeightStr), atom_number(HeightStr, Height),
    readWord(FileStream,DepthStr),  atom_number(DepthStr,  Depth),
    % save metadata
    TotalPixelCount is Width*Height,
    assert(imageMeta(ImageId, Width,Height,Depth,Path,TotalPixelCount)),
    debug(imageLoader,'Loading Pixel Data'),
    % loading image data
    !, loadAssertAllPixels(FileStream,TotalPixelCount,0,ImageId,Width).
```

High Performance Image Loader (3)

```
% stop if pixelcount is length of accumulator
loadAssertAllPixels(_,PixelCount,PixelCount,_,_):-
    debug(v,"Finished loading Pixel Data").
loadAssertAllPixels(FileStream, PixelCount,CurrentPixelCount,ImageId,Width):-
    % while stream contains data read them
    !,readRGBTuple(FileStream,(R,G,B)),
    !,X is mod(CurrentPixelCount,Width),
    !,Y is floor(CurrentPixelCount / Width),
    !,assert(imagePixel(ImageId, X,Y,R,G,B)),
    !,NewPixelCount is CurrentPixelCount + 1,
    !,loadAssertAllPixels(FileStream, PixelCount,NewPixelCount,ImageId,Width).

% reads a rgb tuple from the file stream
readRGBTuple(FileStream,(R,G,B)):-
    !,readWordIgnoreNewLine(FileStream,RStr),!, atom_number(RStr, R),
    !,readWordIgnoreNewLine(FileStream,GStr),!, atom_number(GStr, G),
    !,readWordIgnoreNewLine(FileStream,BStr),!, atom_number(BStr, B).
```

Old DiffList Approach (Slow Random Access)

```
diffToList(DList,List):- dappend(DList,[]-[],List-[]).
addToDiffList(List-[Element| Tail2], Element, List-Tail2).

loadAllPixels(_,PixelData,Acc,PixelCount,PixelCount):-
    diffToList(Acc,AccNewList), PixelData = AccNewList,!.

loadAllPixels(FileStream, Result ,Acc,PixelCount,CurrentPixelCount):-
    !,readRGBTuple(FileStream,(R,G,B)),
    !,addToDiffList(Acc,(R,G,B),NewAcc),
    !,NewPixelCount is CurrentPixelCount + 1,
    !,loadAllPixels(FileStream, Result, NewAcc,PixelCount,NewPixelCount).

processLoadImageStream(FileStream,Path,Image) :-
    ....
    !, loadAllPixels(FileStream,PixelArray,X-X,CorrectPixelCount,0),
    Image = ((Width,Height,Depth,Path),PixelArray),!.
```

High Performance Image Writer (1)

```
% writes a given ImageId into the output folder
writeOutput(ImageId):-
    !,string_concat('./img/output/',ImageId,FilePath),
    !,writeImage(FilePath,ImageId).

% writes the given image to a given path
writeImage(Path,ImageId):-
    setup_call_cleanup(
        (
         debug(imageWriter,'Opening File'),
         open(Path, write, FileStream,[buffer(full),close_on_abort(true)])
         ),
        (!,writeImageStream(FileStream,ImageId),!),
        (debug(imageWriter,'Closing File'),close(FileStream))
    ).
```

High Performance Image Writer (2)

```
% writes the image into the open filestream
writeImageStream(FileStream,ImageId) :-
    !,debug(imageWriter,'Write Header'),
    !,imageMeta(ImageId, Width,Height,Depth,_,TotalPixelCount),
    !,write(FileStream,'P3\n'),
    !,debug(imageWriter,'Write Comment Field'),
    !,write(FileStream,'# Written by Prolog\n'),
    !,debug(imageWriter,'Write Image Spec'),
    !,write(FileStream,Width),
    !,write(FileStream,' '),
    !,write(FileStream,Height),
    !,write(FileStream,'\n'),
    !,write(FileStream,Depth),
    !,write(FileStream,'\n'),
    !,debug(imageWriter,'Writing Pixel Data'),
    !, iterativeWrite(FileStream,ImageId,Width,0,TotalPixelCount).
```

High Performance Image Writer (3)

```
% recursive writing until all images are written
iterativeWrite(_,_,_     ,TotalPixelCount,TotalPixelCount).
iterativeWrite(FileStream,ImageId,Width,Counter,TotalPixelCount):-
        !,CurX is mod(Counter,Width),
        !,CurY is floor(Counter / Width),
        !,NewCounter is Counter + 1,
        !,imagePixel(ImageId, CurX,CurY,R,G,B),
        !,writeRGBTuple(FileStream,(R,G,B)),
        !,iterativeWrite(FileStream,ImageId,Width,NewCounter,TotalPixelCount).

% writes a single rgb image to filestream
writeRGBTuple(FileStream,(R,G,B)):-
    write(FileStream,R),
    write(FileStream," "),
    write(FileStream,G),
    write(FileStream," "),
    write(FileStream,B),
    write(FileStream," \n").
```

Image Handling Util

```
% removes all data from an ImageId from memory
deleteImage(ImageId):-
    retractall(imageMeta( ImageId, _,_,_,_,_)),
    retractall(imagePixel(ImageId, _,_,_,_,_)),
    retractall(imageValue(ImageId, _,_ )).

% reads the 8 next pixels for a given pixel to allow kernel calculations
pixelKernel(ImageId,H,W,RLT,GLT,BLT,RMT,GMT,BMT,RRT,GRT,BRT,RLM,GLM,BLM,RRM,GRM,BRM,RLB,GLB,BLB,RMB,GMB,BMB,RRB,GRB,BRB):-
    imageMeta(ImageId, ImageWidth,ImageHeight,_,_,_),
    WidthMinus is ImageWidth - 1, HeightMinus is ImageHeight - 1,
    H @> 0,W @> 0,H @< HeightMinus,W @< WidthMinus,!,
    HMinus is H - 1,WMinus is W - 1, HPlus  is H + 1,WPlus  is W + 1,
    imagePixel(ImageId,HMinus,WMinus,RLT,GLT,BLT),
    imagePixel(ImageId,HMinus,W     ,RMT,GMT,BMT),
    imagePixel(ImageId,HMinus,WPlus ,RRT,GRT,BRT),
    imagePixel(ImageId,H     ,WMinus,RLM,GLM,BLM),
    imagePixel(ImageId,H     ,WPlus ,RRM,GRM,BRM),
    imagePixel(ImageId,HPlus ,WMinus,RLB,GLB,BLB),
    imagePixel(ImageId,HPlus ,W     ,RMB,GMB,BMB),
    imagePixel(ImageId,HPlus ,WPlus ,RRB,GRB,BRB).
```

Image Value Generation (1)

```
% for a given ImageId, calcualte the values and assert them
calculateImageValues(ImageId):-
    info("Calculate Image Values..."),
    assertImageValuesForImageId(ImageId),
    sobelEdgeDetection(ImageId,SobelImageId),
    assertImageValuesForImageId(SobelImageId),
    imageValue(SobelImageId,avgBrightness,SharpnessFactor),
    assert(imageValue(ImageId,sharpness,SharpnessFactor)),
    deleteImage(SobelImageId).
```

Image Value Generation (2)

```
% asserts all relevant image values for a given ImageId
assertImageValuesForImageId(ImageId):-
    imageMeta(ImageId, _,_,Depth,_,TotalPixelCount),
    % PREPAIR FACTS FOR ITERATION STEP
    debug(imageValues,"Prepare Facts"),
    assert(imageValue(ImageId,minBrightness,Depth)),
    assert(imageValue(ImageId,maxBrightness,0)),
    assert(imageValue(ImageId,totalR,0)),
    assert(imageValue(ImageId,totalG,0)),
    assert(imageValue(ImageId,totalB,0)),
    …
```

Image Value Generation (3)

```
...
% ITERATION STEP #1
!,forall(imagePixel(ImageId,H,W,R,G,B),
     (!,
     % MIN BRIGHTNESS
     imageValue(ImageId,minBrightness,CurMinBrightness),!,
     MinBrightness is floor((R+G+B) / 3),!,
     NewMinBrightness is min(CurMinBrightness,MinBrightness),!,
     retract(imageValue(ImageId,minBrightness,_)),!,
     assert(imageValue(ImageId, minBrightness, NewMinBrightness)),!,
     … (same for max brighness) ...
     % ADD TOTALS
     imageValue(ImageId,totalR,CurR), NewR is CurR + R,!,
     retract(imageValue(ImageId,totalR,_)),!,
     assert(imageValue(ImageId, totalR, NewR)),!,
     … (same for g&b) ...
     )
 ),!,
 …
```

Image Value Generation (4)

```
…
% AVERAGE CALCULATIONS
imageValue(ImageId,totalR,TotalR),
imageValue(ImageId,totalG,TotalG),
imageValue(ImageId,totalB,TotalB),
AvgR is TotalR / TotalPixelCount,
AvgG is TotalG / TotalPixelCount,
AvgB is TotalB / TotalPixelCount,
assert(imageValue(ImageId,avgR,AvgR)),
assert(imageValue(ImageId,avgG,AvgG)),
assert(imageValue(ImageId,avgB,AvgB)),
AvgBrightness is (AvgR+AvgG+AvgB) / 3,
assert(imageValue(ImageId, avgBrightness, AvgBrightness)),
…
```

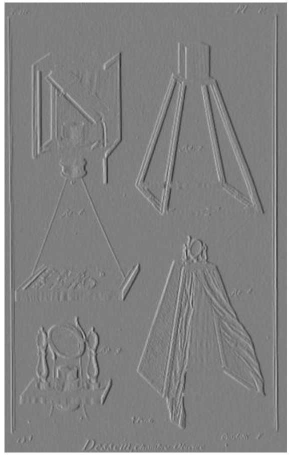
Image Value Generation (5)

```
...
% VARIANCE CALCULATIONS
% calc variance with two-pass algoritm
assert(imageValue(ImageId,stdSumR,0)),
assert(imageValue(ImageId,stdSumG,0)),
assert(imageValue(ImageId,stdSumB,0)),
assert(imageValue(ImageId,stdSumBrightness,0)),
debug(imageValues,"Second Iteration Started"),
!,
forall(imagePixel(ImageId,H,W,R,G,B),(!,
    imageValue(ImageId, stdSumR, StdSumROld),!,
    SdtSumRNew is StdSumROld + ((R-AvgR) * (R-AvgR)),!,
    retract(imageValue(ImageId, stdSumR, _)),!,
    assert(imageValue(ImageId, stdSumR, SdtSumRNew)),!,
     … (same for g&b&brightness) …
)),!,
debug(imageValues,"Second Iteration Done"),
% calc std and variance
imageValue(ImageId, stdSumR, StdSumR),!,
StdR is StdSumR / (TotalPixelCount - 1), VarR is sqrt(StdR),!,
assert(imageValue(ImageId, stdR, StdR )),!,
assert(imageValue(ImageId, varR, VarR )),!.
```

Sharpness via Sobel-Transformation

```
% edge detection with sobel
sobelEdgeDetection(ImageId,SobelImageId) :-
    string_concat(ImageId,".sobel.ppm",SobelImageId),
    copyImageMetaData(ImageId,SobelImageId),
    forall(imagePixel(ImageId,H,W,R,G,B), sobelEdgeDetectionStep(imagePixel(ImageId,H,W,R,G,B),SobelImageId)).

% step that is performed on each pixel, can fail and goto default
sobelEdgeDetectionStep(imagePixel(ImageId,H,W,_,_,_),SobelImageId):-
    imageMeta(ImageId, _,_,Depth,_,_),
    pixelKernel(ImageId,H,W,RLT,GLT,BLT,RMT,GMT,BMT,RRT,GRT,BRT, RLM,GLM,BLM,
                RRM,GRM,BRM,RLB,GLB,BLB,RMB,GMB,BMB,RRB,GRB,BRB),
    LT is RLT+GLT+BLT,!, MT is RMT+GMT+BMT,!,
    RT is RRT+GRT+BRT,!, LM is RLM+GLM+BLM,!,
    RM is RRM+GRM+BRM,!, LB is RLB+GLB+BLB,!,
    MB is RMB+GMB+BMB,!, RB is RRB+GRB+BRB,!,
    Gx is (-LT) + (-MT - MT) + (-RT) + LB + (MB+MB) + RB,
    Gy is (-LT) + RT + (-LM - LM) + (RM+RM) + (-LB) + RB,
    Temp1 is Gx*Gx, Temp2 is Gy*Gy, Temp3 is Temp1 + Temp2,
    Temp4 is sqrt(Temp3), Temp5 is Temp4 / 4328 * Depth,
    Length is floor(Temp5),
    assert(imagePixel(SobelImageId,H,W,Length,Length,Length)).
% we have no fail case here - so we need a fallback for the corner cases
sobelEdgeDetectionStep(imagePixel(_,H,W,_,_,_),SobelImageId) :-
    assert(imagePixel(SobelImageId,H,W,0,0,0)).
```



$$\mathbf{G}_x = \mathbf{S}_x * A = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A$$

$$\mathbf{G}_y = \mathbf{S}_y * A = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Available Values and Features

1. *Maximum Brightness*
   a. **Function:** The whitest Pixel
   b. **Features:** maxBrightness : white_too_dark, white_normal
2. *Minimum Brightness*
   a. **Function:** The blackest Pixel
   b. **Features:** minBrightness : black_too_bright, black_normal
3. *Average Brightness*
   a. **Function:** The Actual Brightness of the Image
   b. **Features:** avgBrightness : too_bright, too_dark, brightness_normal
4. *Standard Deviation of the Brightness*
   a. **Function:** Contrast
   b. **Features:** stdBrightness : high_contrast, normal_contrast, low_contrast
5. *Average Color Vector - RBG*
   a. **Function**: The prevalent color
   b. **Features**: avgR, avgG, avgB: much_red, much_green, much_blue
      avgR :: ~>avgG, ~>avgB, *>100 ---> much_red.
6. *RBG Standard Deviation*
   a. **Function**: The Saturation
   b. **Features**: stdR + stdG + stdB : colorful
7. *Sharpness*
   a. **Function**: Is the image in focus
   b. **Features**: sharp or blurry

Thresholds and Features

Small Language to define different Thresholds for each Attribute:

<attribute> :: *<functor><threshold>[, …] ---> <feature>

Example:

```
brightness :: *>0.7 ---> bright.
brightness :: *>=0.3, *=<0.7 ---> normal_bright.
brightness :: *< 0.3 ---> dark.
```

Thresholds and Features - Update

We added additional functions to the Metainterpreter, so one rule can use multiple values

1. **Addition *+**

   ```
   stdR :: *+stdG, *+stdB, *>15000 ---> colorful.
   ```

2. **Comparing Multiple Values ~>, ~<**

   ```
   avgR :: ~>avgG, ~>avgB, *>100 ---> much_red.
   ```

The Threshold Metainterpreter

```
/* For each relevant value a single feature is extracted  */
extractFeatures([]):-  info("Extracted Features:").
extractFeatures([imageValue(ImageId,Key,Value)|ValueList]) :-
   extractSingleFeature(imageValue(ImageId,Key,Value)),
   extractFeatures(ValueList).
```

The Threshold Metainterpreter

```
/* Each value has a key. For each key there is a threshold
   rule in thresholds.pl. This predicate searches a rule
   identified by the Key of the current value, that can be
   unified with chosen Value. The result is features which
   are immediately added to the Prolog KDB */
extractSingleFeature(imageValue(ImageId,Key,Value)):-
   call(::, Key, (Threshold ---> Feature)),
   applyThreshold(ImageId, Value, Threshold),
   sinfo(Key), sinfo(': '), sinfo(Value), nl,
   assert(feature(ImageId, Feature)).
extractSingleFeature(imageValue(_, _, _)).
```

The Threshold Metainterpreter - Comparing

```
/* These are a couple of operations the threshold rule system is
able to interpret. The simplest operators compare a threshold with
the current value  */
applyThreshold(_, Value,  *<Threshold) :-  Value <  Threshold.
applyThreshold(_, Value, *=<Threshold) :-  Value =< Threshold.
applyThreshold(_, Value,  *>Threshold) :-  Value >  Threshold.
applyThreshold(_, Value, *>=Threshold) :-  Value >= Threshold.
```

The Threshold Metainterpreter - Different Values

```
/* These more complex operators can be used to compare the current
value to another value identified by its Key */
applyThreshold(ImageId, Value,  ~<Key) :-
    imageValue(ImageId, Key, Threshold),
    Value < Threshold.
applyThreshold(ImageId, Value,  ~>Key) :-
    imageValue(ImageId, Key, Threshold),
    Value > Threshold.
```

The Threshold Metainterpreter - Adding

```
/* Other values can be added to the current value. This can be repeated and
   then the sum can be compared to a threshold */
applyThreshold(ImageId, Value, (*+Key, B)) :-
   imageValue(ImageId, Key, AnotherValue),
   NewValue is Value + AnotherValue,
   applyThreshold(ImageId, NewValue, B).

/* A rule can consist of an arbitrary number of conditions */
applyThreshold(ImageId, Value, (A, B)) :-
   applyThreshold(ImageId, Value, A),
   applyThreshold(ImageId, Value, B).
```

Threshold Examples

```
stdBrightness :: *>4500 ---> high_contrast.
stdBrightness :: *>=1500, *=<4500 ---> normal_contrast.
stdBrightness :: *<1500 ---> low_contrast.

avgR :: ~>avgG, ~>avgB, *>100 ---> much_red.
avgG :: ~>avgR, ~>avgB, *>100 ---> much_green.
avgB :: ~>avgR, ~>avgG, *>100 ---> much_blue.

stdR :: *+stdG, *+stdB, *>15000 ---> colorful.
```

Features to Filter and “Learning”

Filter Rules are simple and expressed in the following fashion:

[<feature>, <feature>, …] ---> <filter>

Rules are just called and either resolved or the user is asked to add a new rule. This is achieved by applying a few random filters to the image. The user can then decide which result they prefer and the favorite is added as a new rule.

There are features for which manifestations are required and others that are only added to the feature vector if the threshold is triggered.

Adding new Filterrules

A subset of the available filters is suggested to the user.
For each they are asked whether they like it.
If they do a new rule deciding that filter is added.

[black_normal, white_too_dark, too_bright, ...] ---> addGreen.

If they do not like any, the new rule now decides that no_filter is used:

[black_normal, white_too_dark, too_bright, ...] ---> noFilter.

Filter Rules Implementation

```
% Check whether a Rule already exists for the feature list
   !,(checkRules(ImageId, Filter) ->
       (
           sinfo('Found Existing Rule. Applying'), info(Filter)
       );
       (
           makeNewRule(ImageId, Filter),
           sinfo('Applying'), sinfo(Filter)
       )
   ), nl, nl,
```

Filter Rules - Checking for Rules

```
/* For each image in a list, we search the Prolog KDB for a Rule with the
   feature vector of the current image at its body.
   This is done by calling a rule in the style of "Features ---> Filters".
   If Prolog finds a fitting rule, we can apply the found filters,
   if not a new Rule has to asserted into the database.
   Then if an image with the same feature vector is called, it will find
   the exisiting rule, which can be applied.
*/
checkRules(ImageId,Filter) :-
   call(--->, Features, Filter),
   log(Features), nl,
   verifyList(ImageId, Features).
```

Filter Rules - Verifying the Body

```
/* Checks if each element in a list is a feature fact in the KDB
for the current image id */
verifyList(_ ,[]).
verifyList(ImageId, [H|T]) :-
   feature(ImageId, H),
   verifyList(ImageId,T).
```

Filter Rules - Verifying the Body

```
/* Selects an arbitrary amount of Filters from the list of
available filters */
suggestRandomFilters(Filters, Amount) :-
   findall(X, availableFilter(X), FilterSet),
   random_permutation(FilterSet, Mixed),
   append(Filters, _, Mixed),
   length(Filters, Amount).
```

User Decision

```
/* For each preview filter the user is asked whether they like it.
   If they say yes the recursion is broken, the resulting rule can be asserted
   and the chosen filter can be applied.
   If they say no, they are asked the same for the next filter, until none are left.
   In that case a rule is created that states that for this specific feature set no
   filter is applied. */
userDecision(_, []) :- assert(newFilter(noFilter)).
userDecision(ImageId, [Filter|Filters]) :-
   nl, nl, sinfo('Do you like'), sinfo(Filter), sinfo('? y/n'), nl,
   get_single_char(C),
   (C == 121  ->
       info("Thank You! Introducing a new Rule based on your decision."),
assert(newFilter(Filter));
       userDecision(ImageId, Filters)
   ).
```

Old Image Processing using DiffLists

```
filterGrayscale((Meta,Pixels),(Meta,SWPixels)) :-
        filterGrayscaleImp(Pixels,SWPixels).
filterGzayscaleImp(Pixels,Result) :-
        filterGrayscaleImp(Pixels,Result, X-X).
filterGrayscaleImp([],Result, DiffAcc) :-
        diffToList(DiffAcc,Result).
filterGrayscaleImp([(R,G,B)|T], Result, DiffAcc) :-
        BW is round(0.21 * R + 0.72 * G + 0.07 * B),
        addToDiffList(DiffAcc, (BW,BW,BW), NewDiffAcc),
        filterGrayscaleImp(T, Result, NewDiffAcc).
```

- Recursive Algorithms on Image Arrays

- Problems:
  - Performance -> DiffList needed
  - Recursion not always easy
  - Random Access Algorithms
- Idea / Open Change
  - Use facts for faster random access and mutable pixels

Available Filters using Pixel Assertion

```
availableFilter(filterGrayScale).
availableFilter(filterHighKey).
availableFilter(filterLowKey).
availableFilter(filterAddRed).
availableFilter(filterRemoveRed).
availableFilter(filterAddGreen).
availableFilter(filterRemoveGreen).
availableFilter(filterAddBlue).
availableFilter(filterRemoveBlue).
```

```
availableFilter(filterSharpenHard).
availableFilter(filterSharpenSoft).
availableFilter(filterBlurHard).
availableFilter(filterBlurSoft).
availableFilter(filterAddContrast).
availableFilter(filterRemoveContrast).
availableFilter(filterAddBrightness).
availableFilter(filterRemoveBrightness).
```

Filters Util

```
% makes sure a rgb color is in a valid range (0-255) and rounded
capColor((R,G,B),(NR,NG,NB)) :-
    capRGB(R,NR),
    capRGB(G,NG),
    capRGB(B,NB).

% caps a single color to (0-255) and rounds the value
capRGB(X,Y) :- X =< 255, X >= 0, Y is round(X).
capRGB(X,Y) :- X >= 255, Y = 255.
capRGB(X,Y) :- X =< 0, Y = 0.
```

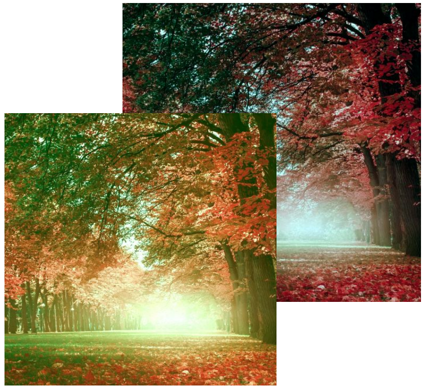
Filters - High and Low Key

```
applyFilter(filterHighKey,ImageId,TargetImageId):-    string_concat(ImageId,".highkey.ppm",TargetImageId),
   implementedFilterMultiply(ImageId,TargetImageId,2,2,2).
applyFilter(filterLowKey,ImageId,TargetImageId):-    string_concat(ImageId,".lowkey.ppm",TargetImageId),
   implementedFilterMultiply(ImageId,TargetImageId,0.5,0.5,0.5).
```

```
implementedFilterMultiply(ImageId,TargetImageId,RMultiple,GMultiple,BMultiple) :-
    copyImageMetaData(ImageId,TargetImageId),
    forall(imagePixel(ImageId,H,W,R,G,B), (
        NR is R*RMultiple, NG is G*GMultiple, NB is B*BMultiple,
        capColor((NR,NG,NB),(TR,TG,TB)),
        assert(imagePixel(TargetImageId,H,W,TR,TG,TB))
    )).
```

Filters - Remove and Add Color

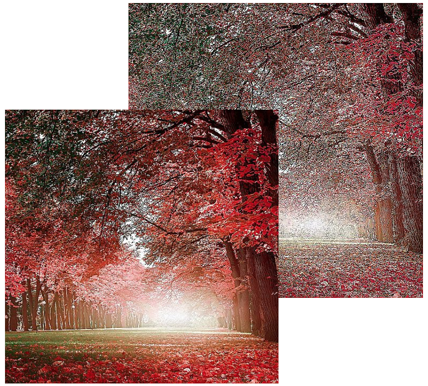
```
implementedFilterAdd(ImageId,TargetImageId,RAdd,GAdd,BAdd) :-
    copyImageMetaData(ImageId,TargetImageId),
    forall(imagePixel(ImageId,H,W,R,G,B), (
        NR is R+RAdd, NG is G+GAdd, NB is B+BAdd,
        capColor((NR,NG,NB),(TR,TG,TB)),
        assert(imagePixel(TargetImageId,H,W,TR,TG,TB))
    )).
```



Filters - Sharpen

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

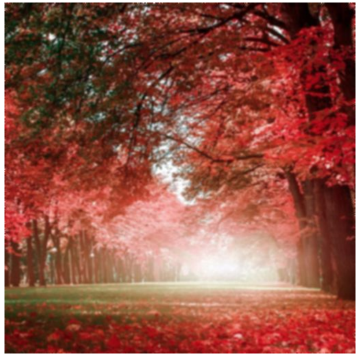
```
implementedFilterSharpen(ImageId,TargetImageId) :-
    copyImageMetaData(ImageId,TargetImageId),
    forall(imagePixel(ImageId,H,W,R,G,B),
        implementedFilterSharpenStep(imagePixel(ImageId,H,W,R,G,B),TargetImageId)
    ).
implementedFilterSharpenStep(imagePixel(ImageId,H,W,R,G,B),TargetImageId):-
    pixelKernel(ImageId,H,W,_,_,_,RMT,GMT,BMT,_,_,_,
                RLM,GLM,BLM,RRM,GRM,BRM,_,_,_,RMB,GMB,BMB,_,_,_),
    SharpenedR = (5*R) - RLM - RRM - RMT - RMB,
    SharpenedG = (5*G) - GLM - GRM - GMT - GMB,
    SharpenedB = (5*B) - BLM - BRM - BMT - BMB,
    capColor((SharpenedR,SharpenedG,SharpenedB),(TR,TG,TB)),
    assert(imagePixel(TargetImageId,H,W,TR,TG,TB)).
implementedFilterSharpenStep(imagePixel(_,H,W,R,G,B),TargetImageId):-
    assert(imagePixel(TargetImageId,H,W,R,G,B)).
```



Filters - Blur

$$\frac{1}{9} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

```
implementedFilterBlur(ImageId,TargetImageId) :-
    copyImageMetaData(ImageId,TargetImageId),
    forall(imagePixel(ImageId,H,W,R,G,B),
        implementedFilterBlurStep(imagePixel(ImageId,H,W,R,G,B),TargetImageId)
    ).
implementedFilterBlurStep(imagePixel(ImageId,H,W,R,G,B),TargetImageId):-
    pixelKernel(ImageId,H,W,RLT,GLT,BLT,RMT,GMT,BMT,RRT,GRT,BRT,
                RLM,GLM,BLM,RRM,GRM,BRM,RLB,GLB,BLB,RMB,GMB,BMB,RRB,GRB,BRB),
    BlurR = (R + RLT + RMT + RRT + RLM + RRM + RLB + RMB + RRB) / 9,
    BlurG = (G + GLT + GMT + GRT + GLM + GRM + GLB + GMB + GRB) / 9,
    BlurB = (B + BLT + BMT + BRT + BLM + BRM + BLB + BMB + BRB) / 9,
    capColor((BlurR,BlurG,BlurB),(TR,TG,TB)),
    assert(imagePixel(TargetImageId,H,W,TR,TG,TB)).
implementedFilterBlurStep(imagePixel(_,H,W,R,G,B),TargetImageId):-
    assert(imagePixel(TargetImageId,H,W,R,G,B)).
```



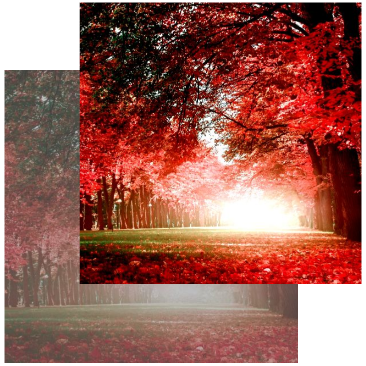
Filters - Gray Scale

```
implementedFilterGrayscale(ImageId,TargetImageId) :-
   copyImageMetaData(ImageId,TargetImageId),
   findall(imagePixel(ImageId,H,W,R,G,B), (
       imagePixel(ImageId,H,W,R,G,B),
       BW is round(0.21 * R + 0.72 * G + 0.07 * B),
       assert(imagePixel(TargetImageId,H,W,BW,BW,BW))
   ),Goals),
   concurrent(2, Goals, []). <- Implemented Concurrent Prolog
```

```
%    the benefit of concurrency is very limited, but it gains some % benefit - also its academically interesting
%    1 thread - % 820,081 inferences, 2.875 CPU in 3.133 seconds (92% CPU, 285246 Lips)
%    1 thread - % 820,081 inferences, 2.656 CPU in 2.989 seconds (89% CPU, 308736 Lips)
%    2 thread - % 2,392,991 inferences, 0.594 CPU in 2.781 seconds (21% CPU, 4030301 Lips)
%    2 thread - % 2,392,991 inferences, 0.547 CPU in 2.743 seconds (20% CPU, 4375755 Lips)
```

Filters - Add and Remove Contrast

```
implementedFilterContrast(ImageId,TargetImageId,Amount) :-
    copyImageMetaData(ImageId,TargetImageId),
    forall(imagePixel(ImageId,H,W,R,G,B), (
        NR is (Amount * (R - 128) + 128),
        NG is (Amount * (G - 128) + 128),
        NB is (Amount * (B - 128) + 128),
        capColor((NR,NG,NB),(TR,TG,TB)),
        assert(imagePixel(TargetImageId,H,W,TR,TG,TB))
    )).
```



Thank you!