Processing Natural Visual Stimuli Using Neural Networks

Thomas Dübendorfer, Kai Jauslin, ETH Zürich
24th March 2000

Dr. Peter König, INI UNI/ETH Zürich
Konrad Körding, INI UNI/ETH Zürich
Dr. Jakob Bernasconi, ETH Zürich

Processing Natural Visual Stimuli Using Neural Networks

Thomas Dübendorfer, Kai Jauslin, ETH Zürich

Supervisors
Dr. Peter König, INI UNI/ETH Zürich
Konrad Körding, INI UNI/ETH Zürich
Dr. Jakob Bernasconi, ETH Zürich

15th May 2000

Contents

1 Natural visual stimuli and neural networks **4**
1.1 Introduction 4

2 Collecting natural visual stimuli **5**
2.1 Sources 5
2.2 Digitizing 5
2.2.1 Procedure 5
2.2.2 Fallacies 6
2.3 Image file formats 6
2.3.1 TIFF (*.TIF) 6
2.3.2 JPEG (*.JPG) 7
2.3.3 AVI (*.AVI) 7
2.4 Archiving 9
2.4.1 desc.mat 9
2.5 Image Gallery 10

3 Preprocessing images for neural networks **13**
3.1 Biological background for preprocessing 13
3.2 Edge detection 14
3.2.1 Canny 15
3.2.2 Evaluation 16
3.2.3 Antagonistic filter 19
3.2.4 Implemented edge detection filters 24
3.3 KUIM Toolkit 24
3.3.1 General 24
3.3.2 Building your own filters 26
3.4 Fallacies 26

4 Introduction to the NVS framework **27**
4.1 General ideas 27
4.2 How to use the framework 28
4.2.1 Control Panel 28
4.2.2 Input settings 28
4.2.3 Network settings 29
4.2.4 Filters 30
4.2.5 Framework 31
4.2.6 Display menu option 31
4.2.7 Starting and stopping the simulation 32

5 Statistics **34**
5.1 Measurements . . . 34
5.2 2D discrete FFT . . . 34
5.3 Improving 2D discrete FFT . . . 34
5.4 Batch statistics using the NVS framework . . . 34
5.5 Results - Plots . . . 37
5.6 Results - Values . . . 39

6 NVS Implementation **41**
6.1 Module overview . . . 41
6.2 Interfaces . . . 41
6.3 Description of global variable structs . . . 42
6.4 Description of individual modules . . . 47
6.4.1 nvs . . . 48
6.4.2 initParameters . . . 48
6.4.3 controlPanel . . . 48
6.4.4 controlPanel_Callback . . . 48
6.4.5 mainLoop . . . 49
6.4.6 initInput . . . 50
6.4.7 initNetwork . . . 51
6.4.8 initFilters . . . 51
6.4.9 chooseImage . . . 51
6.4.10 filterImage . . . 52
6.4.11 calcStatistics . . . 52
6.4.12 setNetworkInput . . . 52
6.4.13 feedForward . . . 52
6.4.14 learn . . . 53
6.4.15 grayscale . . . 53
6.4.16 statistic . . . 54
6.4.17 inputSettings, networkSettings, filterSettings, frameworkSettings . . . 54
6.4.18 drawActivity, drawInput, drawWeightsWinner, showOverallStatistics, showSingleStatistics . . . 54

A Tools **55**
A.1 nameplus.pl . . . 55
A.2 batch.pl . . . 56

B Bibliography **57**
B.1 Image Processing - Mathematics . . . 57
B.2 Image Processing - Biological Background . . . 58
B.3 Image Processing - Neural networks . . . 59

C Acknowledgements **60**

1 Natural visual stimuli and neural networks

1.1 Introduction

Vision research far too often still tries to keep the dirty environment we live in out of the clean labs. It restricts itself to processing mostly synthetic images. However if we want to build smart image processing tools we have to deal with natural visual stimuli instead.

There are many approaches to neural networks that recognize clean synthetic images (e.g. arbitrary oriented bars or filled circles). When working with images of visual stimuli taken directly from the real nature we live in, those methods often cannot be applied anymore as the assumptions made for the 'clean lab' algorithms do no longer hold true.

This semester project tries to transfer a new learning rule from the lab to the real world. We collected representative natural visual stimuli, digitized the material, investigated various algorithms for preprocessing the images, calculated statistics on the data, built a generic NVS (natural visual stimuli) framework with MATLAB to support interactive processing of the images and feeding them to a configurable neural network and finally adopted the unsupervised learning rule "learning with two sites of synaptic integration" (cf. [19]) that was developed at the Institute of Neuroinformatics INI at the University/ETH of Zurich by Konrad Körding and Dr. Peter König.

Chapter 1-3 discuss various aspects concerning image preprocessing of natural visual stimuli for neural networks. Chapter 4 and 6 describe the NVS framework and its implementation. Chapter 5 shows some interesting statistics about the collected natural visual stimuli.

2 Collecting natural visual stimuli

In order to process any natural visual stimuli, we first had to create videos, digitize and convert them to an appropriate file format and provide a convenient way to work with sequences of images.

2.1 Sources

At the beginning of our semester project was the interesting question: "What is a natural visual stimulus?". As the stimulus has to be *visual* we have to look for images as sources. The term *natural* implies that we should definitely not create simple geometric forms like bars or filled circles on a clean background. The easiest solution would therefore be to get a camcorder and record a video of a scene in nature. We ran off to the Irchelpark (a park near the University of Zurich) armed with an analog camcorder and recorded some reed and bubbles. On another day we recorded trees, leaves, sculptures and a chair in a garden covered with snow.
Later we extended the notion of what *natural* should be to anything a human, animal or robot sees during daytime. This includes films on TV (e.g. cartoons) and play tools (e.g. duplos, wooden play tools). We took sequences from a Marsupilami and from a Tom & Jerry cartoon. At the Institute of Neuroinformatics (INI) we found a robot called Khepera that had a colour camera mounted on it. We seized this occasion and grabbed some image sequences from his view point of a world full of play tools.

In the end we had a total of about 55'000 images grouped into three categories (this is about 30 minutes of video at 30 frames per second):

- Nature (Winter, Reed and bubbles)
- Cartoon (Marsupilami, Tom & Jerry)
- Khephera (Wooden play tools, duplos)

2.2 Digitizing

2.2.1 Procedure

The procedure of digitizing analog video material is quite simple. You need an analog/digital video converter card for your computer and a frame grabbing software. Then you connect your camcorder or video recorder directly to your video card and record the video stream. The frame grabbing software we used was Hauppage's WinTV that only supported the AVI video format.
We were primarily interested in having a sequence of images instead of having one large video file. Video files (and especially AVI files) can easily grow

to gigabytes in size. A 320x240 pixel sized video stream with 30 frames per seconds and 24 bits of colours produces about 150 Megabytes of data per minute. Adobe's Premiere video software was used to convert the AVI-files to sequences of image files named as <prefix><sequenceNr>.tif. We chose the TIFF format to not cause any compression errors to the image files so they may be used for further experiments.
Finally as we intended to use the KUIM image processing framework that works on JPEG images we used Paint Shop Pro's batch mode to convert the TIFF images to the JPEG format using the lowest possible compression ratio.
All the digitized image files (AVI, TIFF, JPEG) consumed on the whole about 22 gigabytes of storage. You can imagine what this means for handling purposes. On a standard 10 Mbit/s local Ethernet it takes almost 2.5 hours to transfer the data to another computer. Therefore we burnt some CD-ROMs with the JPEG files on it what allowed us to use the images anywhere. Details about the various image file formats we used can be found at 2.3.

2.2.2 Fallacies

Unfortunately, digitizing is a cumbersome task. First of all you always deal with a huge amount of data that slows all operations down. Secondly there are diverse factors that can badly influence the quality of the digitized images. A mobile phone that goes off during digitization or a computer that is switched on can cause wrong colours, lost synchronization or other nasty effects on the digitized image. Care also has to be taken to choose good quality video plugs (especially when using a T-piece to connect to a separate monitor). Depending on the video card and software it may happen that the last few lines of the digitized image displays patterns of dark bars. This mostly has to do with synchronization problems of your video card.

2.3 Image file formats

2.3.1 TIFF (*.TIF)

TIFF (Tag Image File Format), pronounced "tiff", was originally developed by Aldus Corporation to save images created by scanners, frame grabbers, and photo editing programs. This format has been widely accepted and supported as an image transfer format not tied to specific scanners, printers, or computer display hardware. TIFF is also a popular format for desktop publishing applications. There are several variations of the format, called extensions, so you may have occasional problems opening one from another source. Some versions are compressed using the LZW or other lossless methods. TIFF files support up to 24-bit of colours ([15]).

2.3.2 JPEG (*.JPG)

JPEG is widely used in digital cameras to compress still images. However, it can also be used for movies and some digital still cameras let you capture short video clips in this format. The advantage of this format is that it can be played on many computer systems but it is CPU-intensive.
The JPEG (Joint Photographic Experts Group) format, pronounced "jay-peg", is by far the most popular format for the display of photographic images on the Web. The term "JPEG" is often used to describe the JFIF file format (JPEG File Interchange Format). JFIF is the actual file format that contains an image compressed with the JPEG method. These newer JFIF files originally used the JPG extension, however, the latest standard calls for using a JIF extension instead. The format is optimized for the display of photographs and doesn't work as well as GIF for type or line drawings (GIF is optimized for those). JPEG images have two distinctive features:

- JPEG uses a lossy compression scheme but you can vary the amount of compression and hence trade off file size for image quality, even making extremely small files with poor quality.
- JPEG supports 24-bit of colours. GIF, the other format widely used on the Web supports only 8-bits.

Compression is performed on blocks of pixels, eight on a side. You can see these blocks when you use the highest levels of compression or greatly enlarge the image. JPEG is a two pass compression and de-compression algorithm. This means it takes longer to load and display than a GIF file. You can save images in a progressive JPEG format that works somewhat like an interlaced GIF. While a standard JPEG loads from top to bottom, a progressive JPEG displays the entire image starting with the largest blocks. This allows the image to be displayed first in low-resolution and then filled in as more data arrives. When you save an image in this format, you can specify the number of progressive scans. Don't use JPEG to save original images you expect to modify later. Every time you open one of these files, and then save it again, the image is compressed. As you go through a series of saves, the image becomes more and more degraded. Be sure to save your originals is a loss-free format such as TIFF or BMP at maximum colour depth. Also, when you save an image as a JPEG, the image on the screen won't reflect the compression unless you load the saved version ([16]).

2.3.3 AVI (*.AVI)

One of the most popular formats for presenting video is Microsoft's AVI format. This format was developed to play videos in the Windows environment. The ubiquituousness of the format has led to the development of a

vast array of resources to play, edit and capture video in AVI format. One of the best things about AVI is this level of support.
The AVI format was developed by Microsoft as part of Video for Windows. AVI stands for Audio Video Interleaved. The drivers and player come with Microsoft Windows '95/'98 and NT, if you have not got them because you use OS/2 or an Apple Macintosh you can get other drivers easily. It is a special case of the RIFF (Resource Interchange File Format). RIFF is a clone of the IFF format invented by Electronic Arts in 1984. It was invented for Deluxe Paint on the Amiga, and when Deluxe Paint went to the PC, so did IFF.
With software alone AVIs will play full motion video and audio in a small window at about 15 frames per second. AVI uses a number of different codecs. There were originally two codecs, Video 1 and RLE (run length encoding), other codecs have been developed by third parties such as Indeo and Cinepak. These codecs offer Mac compatibility and audio compression ([17]).
Advantages of AVI:

- AVI has a wide range of video qualities. It can exist in 256 to Millions of colours encoding as well as support sound from 5kHz Mono to CD quality stereo sound. Of course the payoff of using a small number of colours and low quality sound is better MB/sec ratio. If you use the cheapest rates available, AVI can deliver video at ratios as low as 0.03MB/sec. However if you use maximum settings, it can run as high as 0.3 MB/sec. Depending on your needs in terms of quality and hard drive space, AVI has a compromise for you.

- AVI and the media player come with Windows, so no drivers need to be obtained. The Indeo drivers for the media player are better for faster machines and will improve quality.

- AVI is a popular standard, many videos have been produced in the format because of it's non requirement of drivers.

- The quality of AVI files with good drivers and good hardware can be quite impressive.

Disadvantages of AVI:

- Like most video formats, AVI can really take a bite out of your hard-drive, if you want video of exceptional quality, be prepared to pay. AVI can also be a little choppy if not encoded or decoded correctly. Faster processors and special graphics accelerators help with this process.

- AVI's are no longer developed by Microsoft, they have left it to third parties for further development while they concentrate on other things.

Directory	Description
Cartoons	
/t000203a/intro/intro*	Tom & Jerry movie - Intro
/t000203b/house/house*	Tom & Jerry movie - Scene with the demolition of a house
/t000203c/money/money*	Tom & Jerry movie - Scene about money
/t000203d/pictures/pictures*	Tom & Jerry movie - Scene where pictures are looked at
/t000124c/marsu/marsu*	Marsupilami Cartoon - A bath for Maurice (1st part)
/t000124d/water/water*	Marsupilami Cartoon - A bath for Maurice (2nd part)
Nature scenes	
/t000124a/winter/winter*	Garden and trees covered with snow
/t000124b/winter/lion*	Lion sculpture, tree and a chair in a garden covered with snow
/t000209a/bubbles/bubbles*	Reed and bubbles (Irchelpark)
Khepera	
/t000124e/khepera/frame*	Coloured wooden play tools recorded from Kheperas perspective
/t000124f/bubbles/bubbles*	Coloured plastic duplos recorded from Kheperas perspective

Table 1: Archive with digitized images

This could be good, but I think it will be bad because the third parties have not got Microsoft's budget, so they won't want to take any risks.

2.4 Archiving

The TIFF images are stored in the path as stated in the table. The JPEG version is in the `/t000*/*_jpg/...` directory. All sequence numbers have size 4, padded with leading zeros. The images are of size 320x240 pixels with 24 bit of colours and were grabbed at 30 frames per second.

2.4.1 desc.mat

In order to provide a convenient way to process the digitized images we provide a description file for each sequence of natural visual stimuli. The file resides in the root folder of a sequence (e.g. `/t000124a/desc.mat`). The file is called desc.mat and is a MATLAB binary file that can be read with MATLAB's load command and defines a variable subspace `V_*` with information about the images.

Name	Description	Class
V_author	Name of the author	char array
V_date	Date of creation	char array
V_desc	[First Last Type]: Type of (sub)sequence(s)	double array
V_files	Subdirectory name and basename (file prefix)	struct array
V_sourcedesc	Details about image source	struct array

Table 2: variables in desc.mat (image sequence description file for MATLAB)

Name		value
V_author	=	Thomas Duebendorfer
V_date	=	24-Jan-2000
V_desc	=	[0 4616 1]
V_files.prim_dirname	=	'winter'
V_files.prim_basename	=	'lion'
V_sourcedesc.desc	=	Garden and trees covered with snow
V_sourcedesc.general	=	Images created from a video for the semester project "Processing natural visual stimuli using neural networks" by Kai Jauslin and Thomas Duebendorfer

Table 3: Sample variables in desc.mat

In MATLAB you get access to the variables stored in desc.mat by changing into the directory where desc.mat resides and then executing 'load desc.mat'. The command 'whos' lists you all currently set variables in the MATLAB workspace. V_desc can hold arbitrary types. We only use type 1 to indicate that the images with sequence numbers from 'first' to 'last' exist in the subdirectory 'V_files.prim_dirname' and have prefix 'V_files.prim_dirname'. They all have a unique number in their name, e.g. lion0123.jpg. All images that we digitized have size 320x240 pixel and 16 bit of colours. This can also be determined when reading an image in MATLAB using the command 'imread'.

2.5 Image Gallery

The following figures show some representative images of the archive we created.

Reed And Bubbles

/t000209a/bubbles/bubbles*.jpg 0000 - 9062

Winter

/t000124a/winter/winter*.jpg 0000 - 4474
/t000124b/winter/lion*.jpg 0000 - 4616

Figure 1: Nature images

Tom & Jerry

/t000203a/intro/intro*.jpg 0000 - 0124
/t000203b/house/house*.jpg 0000 - 4892
/t000203c/money/money*.jpg 0000 - 1240
/t000203d/pictures/pictures*.jpg 0000 - 3281

Marsupilami - a bath for Maurice

/t000124c/marsu/marsu*.jpg 0000 - 4194
/t000124d/water/water*.jpg 0000 - 5408

Figure 2: Cartoon images

Figure 3: Khepera images

3 Preprocessing images for neural networks

3.1 Biological background for preprocessing

David H. Hubel describes in [10] how a human reacts to visual stimuli. He explains that even a single photon can be noticed by a cell in the retina and transformed into an electric signal. This signal then travels through various types of cells and is aggregated several times. Some retina cells, the lateral geniculate nucleus (one on each side), and the visual cortex seem to do some kind of edge detection.

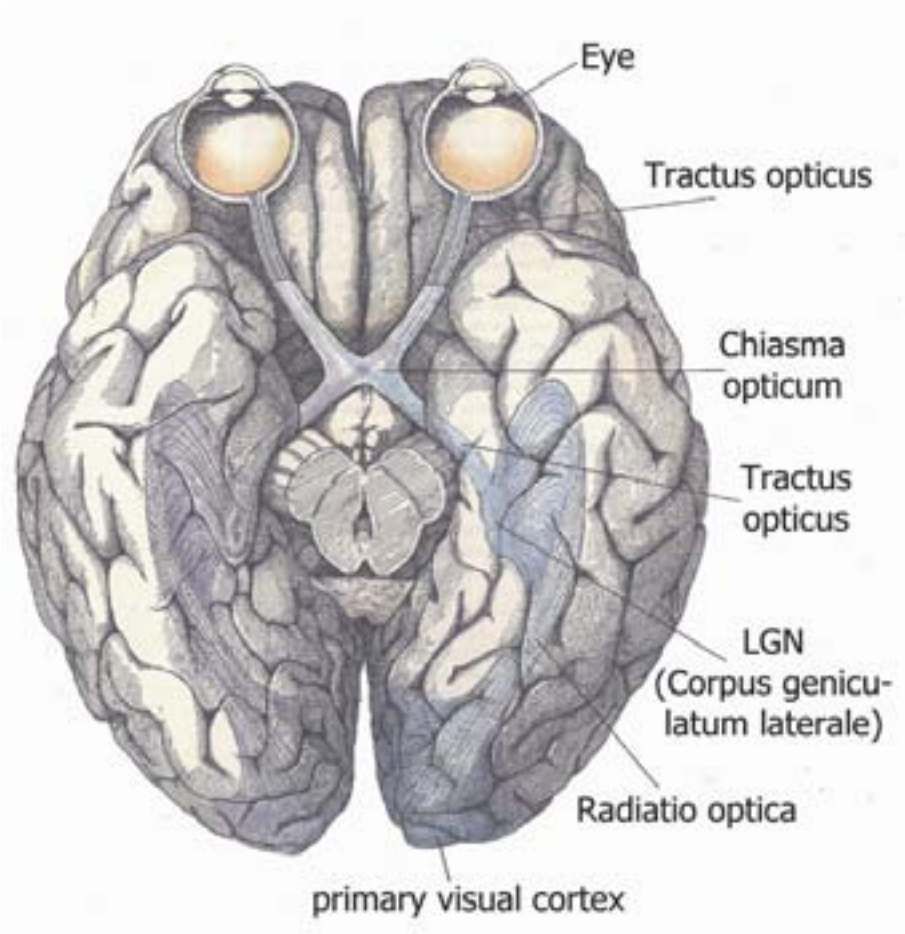


Figure 4: The process of seeing (image source: [10])

An interesting fact is that the worst visual stimulus is a constant light beam like a pocket lamp provides it. The best one is a very small light beam that switches on and off. The reason is that the cells are connected such that they react excitatory to light in the center and inhibitatory to light outside the

center (and vice versa for other cell types). This allows the brain to connect certain nerves to get the information "there is a line oriented in that way" as figure 5 shows. This is also the reason why we will investigate some edge detection algorithms and apply them for image preprocessing. The neural network we want to feed the processed images to is located somewhere in the visual cortex, so the signal was already "edge detected' on its way to the network.
Information specific to the cat's visual stimuli processing (which is very similar to the human's processing) can be found in [11].

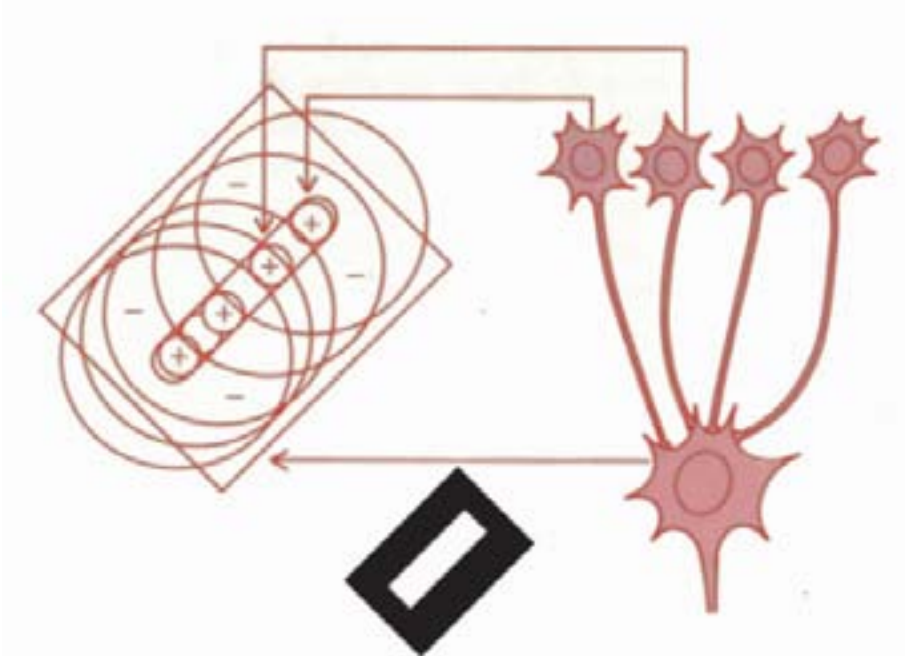


Figure 5: Edge detection by aggregation of neural signal.

3.2 Edge detection

Now we turn our attention to edge detection image filters. When we consult some literature (e.g. [1]) about image processing we find many aproaches that deal with edge detection. It seems to be very hard to create an algorithm that works for all sorts of images equally well. Neither there is consensus on what means that an edge detector works well.
In 1996 a group of the Computer Science & Engineering Department of Psychology at the University of South Florida made an attempt on finding out which edge detection algorithm is superior to others on different images (cf. [3]). In a pragmatic way they presented a set of edge detected images to

different students. They had to rate which of several edge detection algorithms produced the most useful output. Useful means that the spectator recognizes as much information as possible in the edge detected image.
The choice of humans to rate it instead of using statistical properties for algorithm rating is easy to understand: Who can do better interpretation of images than a human?

3.2.1 Canny

Canny is one of the most common edge detection filters. It is described by its inventor in [4] so we will not explain it in detail here.

Sometimes the images are filtered with canny (which produces an image with many grey pixels) and thereafter a "zeros crossing" called method tries to turn the picture into a black/white image and closed curves. Unfortunately the "zeros crossing" step can produce bad results if applied to natural visual stimuli (cf. image bubble2283.jpg).

Zeros (zeros crossing on canny filter) looks as follows:

1. Smooth the image using the Gaussian filter with some fixed width
2. compute local gradient G
3. compute second directional derivative D in the direction of local gradient
4. identify zero crossings of D, that is, closed contours defined by D = 0 and
5. accept or reject the resulting edges based on some signal-to-noise evaluation technique.

Other widely used edge detectors (such as Sobel, finite difference and cubic spline) rely on derivation.

Many neural network's learning rules have severe difficulties learning with input stimuli that have values close together (i.e. if all are gray and there are no strong differences to lighter or darker pixels). Zeros crossing is one method to help but unfortunately for some types of images this method is not suited at all. And one of this unsuited types of images is scenes from nature. We therefore created an own algorithm called antagonist (cf. 3.2.3) that adapts to the input data and can also deal with natural visual stimuli.

3.2.2 Evaluation

The following figures show a selection of edge detection algorithms applied to natural visual stimuli. The original was grayscaled and then processed separatedly with canny and cubic spline derivation. The bottom row shows the antagonistic filter (cf. 3.2.3) and zeros crossing applied to the edge detected image.

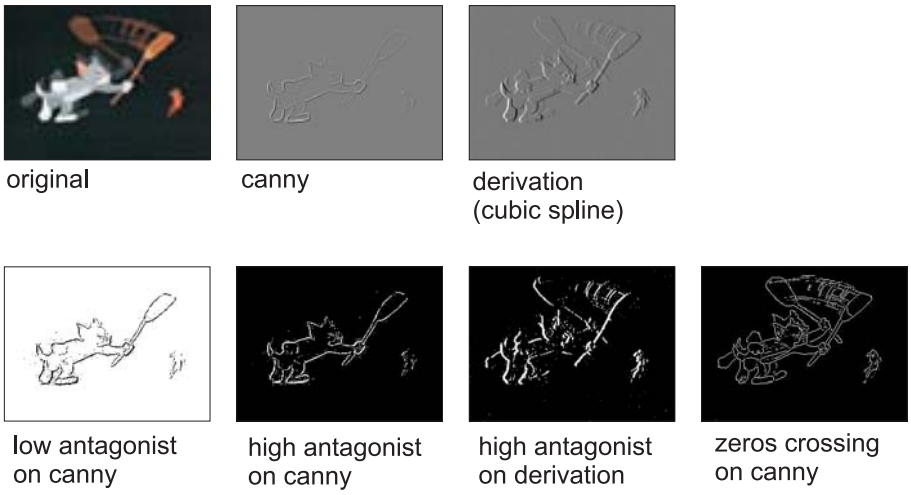


Figure 6: Tom & Jerry Intro - `intro0063.jpg`

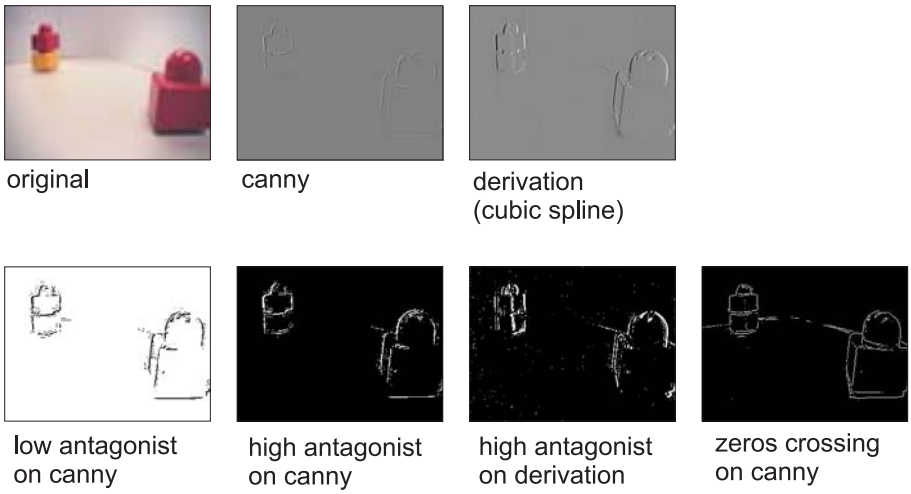


Figure 7: Khepera duplos - `duplo8219.jpg`

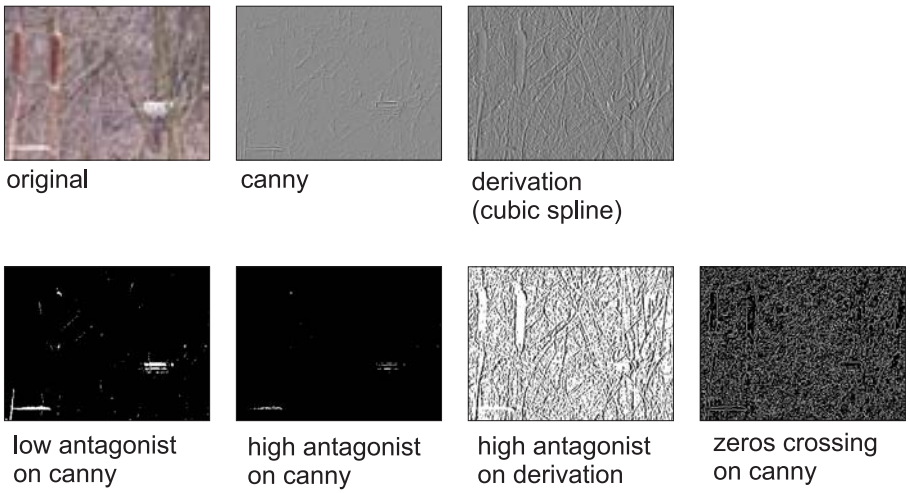


Figure 8: Reed and bubbles - `bubble2283.jpg`

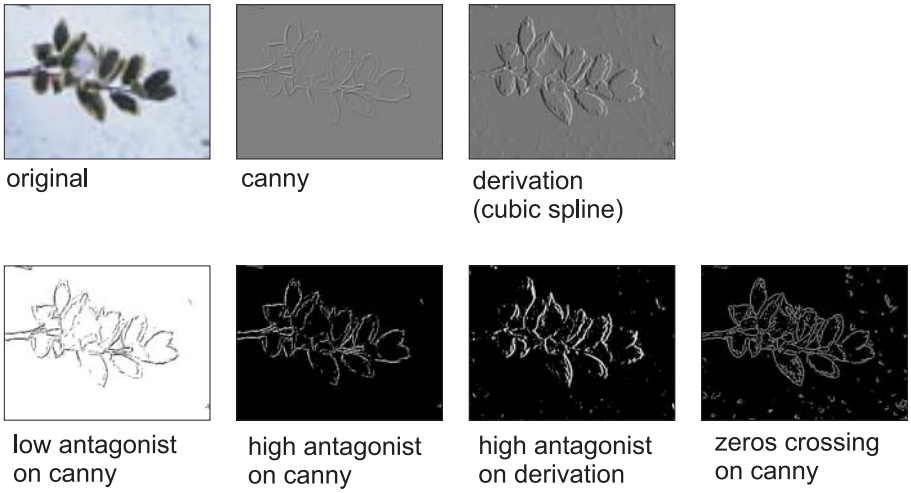


Figure 9: Winter - `lion4439.jpg`

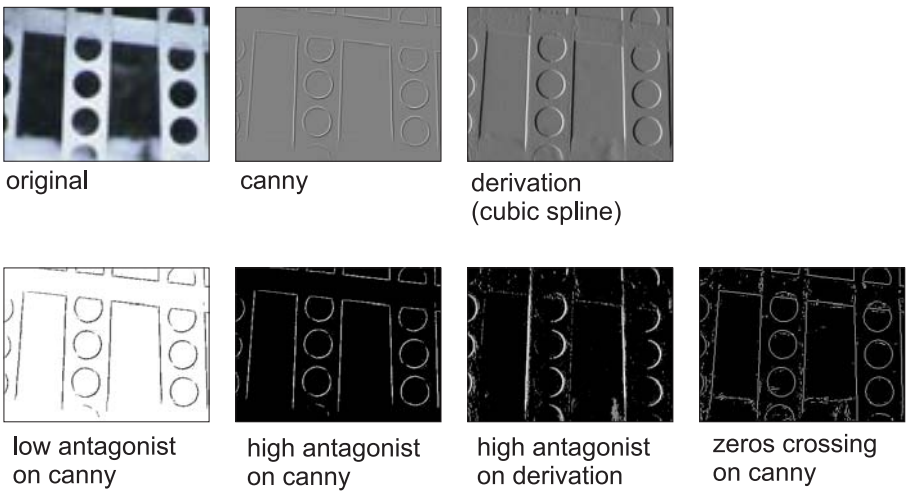


Figure 10: Winter - `lion4126.jpg`

3.2.3 Antagonistic filter

We already explained why there is a need to further process edge detected images (cf. 3.2.1). The aim of the development of the antagonistic filter was to optimize edge detected images as input into neural networks.

Before we show the MATLAB code of the antagonistic filter we want to outline the ideas behind.

The interface looks like this:
`[imgA,split,iMin,iMax] = antagonist(img, range, part, fThresh);`
We provide an image <img>, a selection <range> of the lower or upper intensity range, an intensity value part <part>, as well as a frequency threshold <fThresh>.

The algorithm works like this: We grayscale the image <img> if not already done and calculate the histogram on the intensity values in the range 0 (black) .. 255 (white). This allows us to find the most frequent intensity, called 'split'. It further helps us to determine the smallest intensity 'iMin' and the highest intensity 'iMax' that are not less frequent than the threshold 'fThresh'.
Now, let us assume the high (resp. low) range was selected. We then calculate how far we have to shift the intensity value iMax (resp. iMin) to reach the maximum (resp. minimum) intensity allowed. Then we select all pixels that are in the part % of the upper (resp. lower) range of iMax minus split (resp. split minus iMin). The intensity of those selected pixels is first shifted (as determined just before) and then all selected pixels are written into the newly created image at their corresponding places. As we use a trick to shift efficiently we have to clip the image at the intensity boundaries to stay inside legal intensity values.

Antagonistic filter - source code

```
function [imgAntago,split,iMin,iMax] = antagonist(img,range,part,fThresh)
% (c) Thomas Duebendorfer, Februrare 2000, INI University/ETH Zurich
%
%   e.g. [imgAntago,split,iMin,iMax] = antagonist(img,1,87,10);
%        get antagonistic image in high range of image img, using 87% of high
%        intensity range and an intensity frequency threshold of 10 pixels
%
% IN:
% img       MxN grayscale image (with intensity values in range [0,1])
% range     extraction range: 0 (low), 1 (high)
% part      percentage of intensity range between split point and boundary
%           (iMax for mode = 0 resp. iMin for mode = 1) in range [1,99]
% fThresh   threshold: minimum frequency of an intensity to be accepted
```

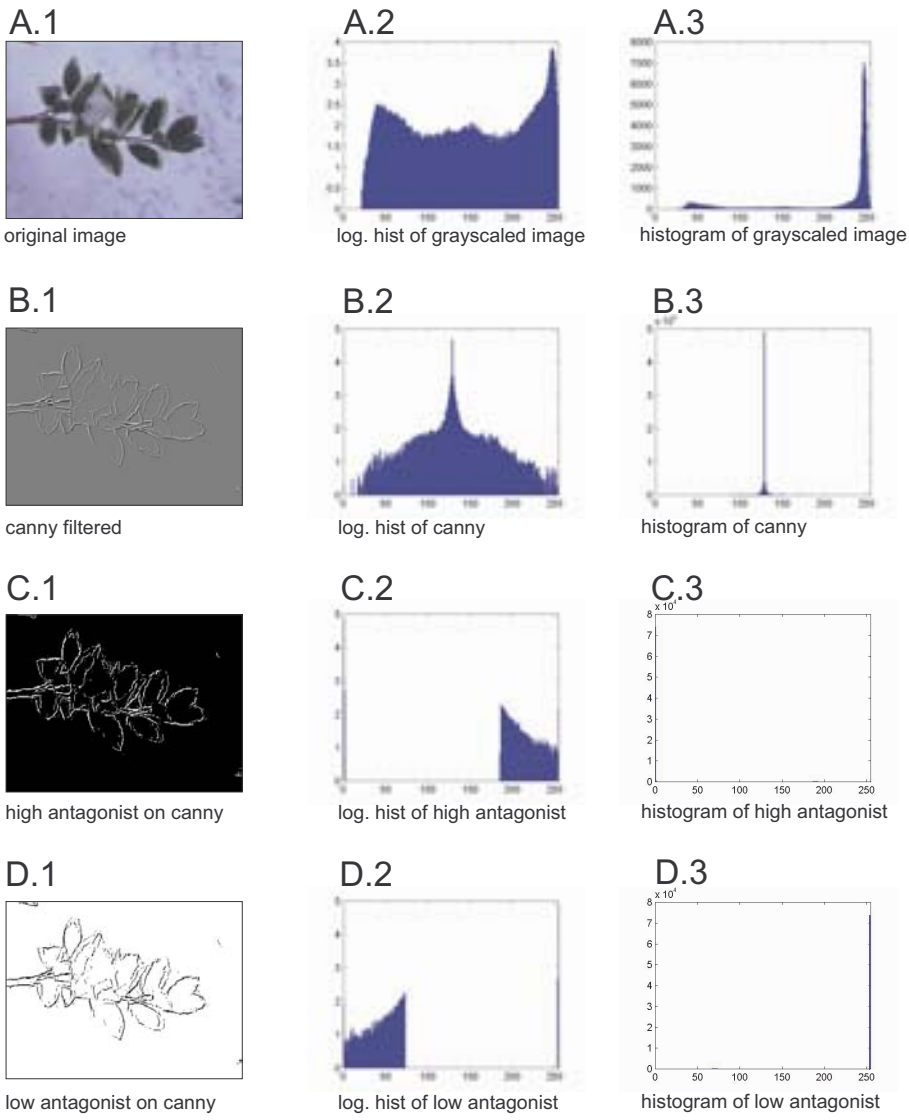


Figure 11: Histograms to illustrate the antagonistic filter.

```
%           as iMin or iMax candidate (in pixels)
%           note: as a rule of thumb holds: the higher the fThres the
%           lower the part should be chosen
%
% RETURN:
% imgAntago is the antagonistic MxN grayscale image of img (with intensity
%           values in range [0,1])
% split     automatically determined split intensity value
% iMin      minimum intensity value that has a frequency >= fThreshold
% iMax      maximum intensity value that has a frequency >= fThreshold
%
% Purpose:  Perform 'antagonistic' filtering on a grayscale image that
%           was processed with an edge detection filter (preferably canny).
%           If finds the most frequent intensity value 'split', the
%           minimum and maximum intensity values iMin and iMax that are
%           more or equal frequent than the threshold fThresh.
%           If the range is 'low' then a black image is created and all
%           pixels with intensity values in the upper 'part' % of
%           the range between split and iMax are inserted after having
%           shifted this part to max. intensity.
%           If the range is 'low' then a white image is created and all
%           pixels with intensity values in the lower 'part' % of
%           the range between iMin and split are inserted after having
%           shifted this part to min. intensity.
%
%           The results of the filter when used for images that were
%           processed by an edge detector before (as canny) are parti-
%           cularly suited to be feeded to neural networks.

% rescale intensity range from real [0,1] to integers [0,255]
if (isa(img,'double'))
     imgAntagoTmp = floor(grayscale(img) * 255);
else
     imgAntagoTmp = double(grayscale(img));
end

% check if part is a reasonable percentage
if (part <= 1)
     part = 1;
elsif (part >= 99)
     part = 99;
end

% find split intensity value automatically
% create histogram of img, [0:255] is number of bins and
% their location returns frequency counts in iFreq,
% bin locations in binLoc
[iFreq,binLoc]     = hist(imgAntagoTmp(:),[0:255]);

% find max value and corresponding index (index range 0..255)
[c,i]     = max(iFreq);


% most frequent intensity value (we will use it as the
```

```
% intensity split point)
split     = binLoc(i);

% number of pixels that have split as their value
num_split = iFreq(i);

% find minium intensity satisfying frequency threshold
iMinIdx = min(find(iFreq >= fThresh));
if isempty(iMinIdx)
  iMin = 0;
else
  % convert index to intensity
  iMin = binLoc(iMinIdx);
end

% find maximum intensity satisfying frequency threshold
iMaxIdx = max(find(iFreq >= fThresh));
if isempty(iMaxIdx)
  iMax = 255;
else
  % convert index to intensity
  iMax = binLoc(iMaxIdx);
end

% ensure iMax and iMin to be in [0,255]
if (iMax > 255)
     iMax = 255;
end
if (iMin < 0)
     iMin = 0;
end

if (range == 0)
  % low range
  % shift and range
  lowShift = iMin;
  lowRange = split - iMin;

  % boundary
  iBMin = iMin + part / 100.0 * lowRange;

  % shift selected dark intensity values and insert them into
  % a new white imag. Note: We shift intensity values smaller
  % than iMin (that are less frequent than fThresh) to negative
  % values. Clipping adjusts this later.
  imgAntago = 255 * ones(size(imgAntagoTmp));
  selectedIdxSet = find(imgAntagoTmp <= iBMin);
  imgAntago(selectedIdxSet) = imgAntagoTmp(selectedIdxSet) - lowShift;

else

  % high range
  % calculate shifts and ranges
  highShift = 255 - iMax;
```

```
  highRange = iMax - split;

  % boundary
  iBMax = iMax - part / 100.0 * highRange;

  % shift selected light intensity values and insert them into
  % a new black imag. Note: We shift intensity values bigger
  % than iMax (that are less frequent than fThresh) to values
  % bigger than 255. Clipping adjusts this later.

  imgAntago      = zeros(size(imgAntagoTmp));
  selectedIdxSet = find(imgAntagoTmp >= iBMax);
  imgAntago(selectedIdxSet) = imgAntagoTmp(selectedIdxSet) + highShift;

end

% clip to [0,255]
% values < 0 become 0, values > 255 become 255
imgAntago(find(imgAntago > 255)) = 255;
imgAntago(find(imgAntago < 0))   = 0;

% recalc intensity values to range [0,1]
imgAntago = double(imgAntago) / 255.0;
```

The file antagonistCannyHigh.m (3.2.3) implements the interface (cf. 3.2.4) for the NVS framework and uses a fixed set of parameters for antagonist.m (and calls canny). This allows to flexibly integrate new filters based on antagonist into the NVS framework.

```
function [] = antagonistCannyHigh(imgIn,imgOut,tmpPath)
% antagonistic filtering (wrapper)
% IN:
% imgIn   path to input  image file
% imgOut  path to output image file
% tmpPath path to temporary directory

% do canny first
tmpCanny = sprintf('%s.canny.jpg',imgOut);
canny = sprintf('!./filters/canny.pl %s %s %s',imgIn, tmpCanny, tmpPath);
eval(canny);

% remove imgOut
rmOut = sprintf('!rm -f %s',imgOut);
eval(rmOut);

% read canny output and apply antagonistic filter
img = imread(tmpCanny);
[imgAntago,split,iMin,iMax] = antagonist(img,1,87,10);
imwrite(imgAntago,imgOut,'jpg','Quality',100);
```

3.2.4 Implemented edge detection filters

Interface:
The `.m` (Matlab) and `.pl`-files (Perl files) are just wrappers on MATLAB m-files, KUIM and UNIX system binaries. The command line calling interface is as simple as: `filter.pl <inImg.jpg> <outImg.jpg> <tmpPath>`, e.g. `canny.pl myInputImg.jpg myOutputImg.Canny.jpg /tmp`.
Some filters require more than one step. Most KUIM filters produce their output as IM-Files (a KUIM internal image file format) that has to be converted to the JPEG format (using the KUIM 'type' tool). The MATLAB calling interface is straight forward too, e.g.:
`antagonist('myInputImg.jpg','myOutputImg.jpg','/tmp')`.
There is no return value as the output image is written as a side effect of calling the function.

`Rangecalc` was written to do various useful calculations on the intensity range of an image (extracting, windowing, histogram equalization and trimming, smart splitting). It lead us to the development of the antagonistic filter.

For further experiments there is also a gaussian blur filter (`gauss.m`) included. Note that all antagonistic filters of table 4 use the core algorithm implemented in `antagonist.m`. The perl scripts call KUIM binaries 'canny', 'rangecalc', 'deriv', 'type' and jpeg tools such as 'cjpeg' (conversion to jpeg), 'jpegtran' (translations of jpeg images, e.g. grayscale). The temporary images are removed before writing to a destination using the UNIX command 'rm'.

3.3 KUIM Toolkit

3.3.1 General

If you are not using Matlab to implement filters (e.g. `rangecalc`) you can use the KUIM Toolkit [9] and write your filters in C. This is fast and many libary functions (including reading and writing JPEGs) are provided.

The KUIM image processing system was implemented at the University of Kansas to support teaching and basic image processing and computer vision research. The image file format and I/O libraries have been designed for ease of use and portability to a wide variety of machines.

The KUIM system consists of over 100 general purpose and special purpose programs implemented in C. In addition, there are Java based graphical user interfaces for roughly 50 of these image processing programs which enable the user to adjust input parameters and see their effects on corresponding

Filter name	Description	Based on
canny.pl	canny edge detection	KUIM
zeros.pl	1. canny 2. zeros crossing	KUIM (canny)
antagonistHigh.m	high antagonist	MATLAB (antagonist)
antagonistHighDetailed.m	high detailed antagonist	MATLAB (antagonist)
antagonistLow.m	low antagonist	MATLAB (antagonist)
antagonistLowDetailed.m	low detailed antagonist	MATLAB (antagonist)
antagonistCannyHigh.m	1. canny 2. high antagonist	KUIM (canny), MATLAB (antagonist)
antagonistCannyHighDetailed.m	1. canny 2. high detailed antagonist	KUIM (canny), MATLAB (antagonist)
antagonistCannyLow.m	1. canny 2. low antagonist	KUIM (canny), MATLAB (antagonist)
antagonistCannyLowDetailed.m	1. canny 2. low detailed antagonist	KUIM (canny), MATLAB (antagonist)
rangecalcCannyHigh.pl	1. canny 2. high rangecalc	KUIM (rangecalc, canny)
rangecalcCannyLow.pl	1. canny 2. low rangecalc	KUIM (rangecalc, canny)
rangecalcHigh.pl	high rangecalc	KUIM (rangecalc, canny)
rangecalcLow.pl	low rangecalc	KUIM (rangecalc, canny)
derivCubicSpline.pl	derivation (cubic spline operator)	KUIM (deriv)
derivFinDiff.pl	derivation (finite differences)	KUIM (deriv)
derivRoberts.pl	derivation (Robert's operator)	KUIM (deriv)
derivSobel.pl	derivation (Sobel's operator)	KUIM (deriv)

Table 4: Implemented edge detection filters for the NVS framework

output images.

The KUIM system is similar in spirit to an earlier package called /usr/image which originated at the University of North Carolina at Chapel Hill in the 1980's under the direction of Prof. Stephen Pizer (smp@cs.unc.edu). Questions about KUIM and suggestions for improvement should be sent to Prof. John Gauch (jgauch@ittc.ukans.edu).

3.3.2 Building your own filters

The list shows how to compile the file "rangecalc.c" with KUIM.
We assume that you have installed the free KUIM Toolkit (c.f. [9]) to /home/myhome/kuim and that your filter source is named rangecalc.c and resides in the path /home/myhome/kuim/rangecalc/rangecalc.c.

1. `> tcsh`
2. `> setenv CC gcc`
3. `> setenv ARCH i686  /* see uname -m for i686 or i586 */`
4. `> setenv KUIM /home/myhome/kuim`
5. `> cd /home//myhome/kuim/src/rangecalc`
6. `> make`

All makefiles used by KUIM filters look almost identical. Only the destination filename has to be adjusted before compiling.

Remark: ARCH is now your machine architecture and KUIM is the path to you KUIM Toolkit directory.

3.4 Fallacies

There is only one thing really worth mentioning. When working with MATLAB you can easily convert a matrix A to a vector by using the :-Operator. A(:) denotes the vector of matrix A in column first order. This means that A(:) == [A(1,1); A(2,1); A(3,1); ... A(1,2); A(2,2); ...]. When working with the KUIM toolkit the image data is in a vector that uses row first order, i.e. A(:) = [A(1,1); A(1,2) ...; A(2,1); A(2,2) ...]. This has to be remembered when porting algorithms from MATLAB to KUIM in order to making them not just faster but also keeping them correct.

4 Introduction to the NVS framework

When processing input data from external sources there are always similar problems – how to read it in and how to connect the algorithms. For this semester project, we decided to develop an interactive framework so that users with only limited knowledge of Matlab would nevertheless be able to work with it and get results. Because of the properties of the images in our collection, it is called "NVS", standing for "Natural visual stimuli". It is possible to use any image as input, natural or not.

4.1 General ideas

The goal of our semester project was to develop a neural network simulation, which can be fed with the collected natural stimuli. The NVS framework extends this idea by offering the user a graphical user interface (GUI) and by encapsulating functionality into small, manageable modules. The executing part of the simulation is separated – as far as possible – from the user interface. The implemented NVS framework functionality of the real important parts consists of:

- Read in collection of input images (JPG or TIFF), recognize desc.mat
- Choose input views
- Use multiple filters
- Build network
- Feed network
- Learn with two sites of synaptic integration
- Calculate statistics

and for the GUI

- New, Open and Save whole simulations, Load and Save settings
- Display progress and simulation state
- Overall (mean) statistics output
- Single image statistics output
- Network activity output
- Winner weights output
- Multiple dialog boxes for simulation settings and display options

4.2 How to use the framework

This section gives you a short introduction on how to use the framework as a normal user. If you are interested in implementational details, please read chapter 6 on the internas.

4.2.1 Control Panel

A graphical NVS Session can be started by first changing into the appropriate directory and typing `nvs` at the Matlab prompt. The first window appearing is the Control Panel.

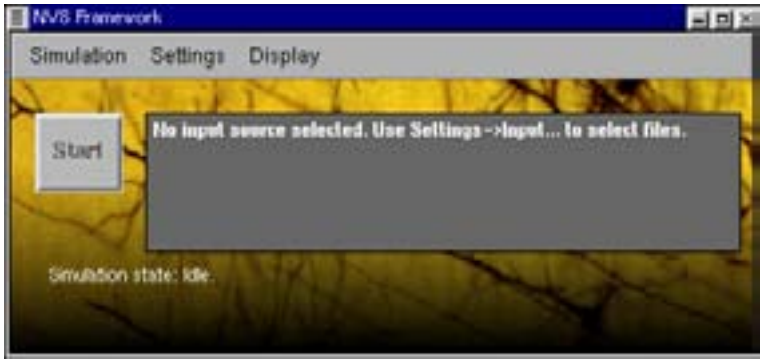


Figure 12: The NVS Control Panel

At the moment it is not possible to use a text only Matlab command line interface. However, most modules do not use any graphical output and can be called while the simulation is not running.

4.2.2 Input settings

When creating a new simulation, it is advisable to start with the selection of your input files. Please use the Settings → Input menu to open the input dialog box (see below).
First, select your files (Choose...) You can choose a Matlab description file (desc.mat) or a individual directory. The format of the desc.mat-File is illustrated in 2.4. To select a directory, choose any file within and click Ok. Next, you have to decide how the input layers of the network should be mapped. For the network model which is used in our project, each of the 2 input layers is about 1/10 up to 1/5 the size of the original image. Thus, overlapping can occur. You can select the maximum overlapping percentage. Last, it is possible to use the same input image for more than one input set. Each input set uses a different mapping.

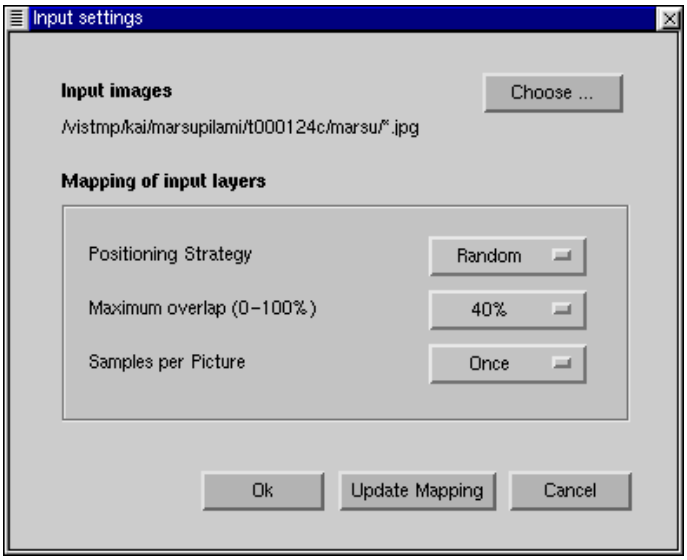


Figure 13: NVS Input settings

When you click the Ok button now, all filenames are read into memory and internal structures concerning the input are initialized. This can be quite a slow process. If you only wish to update the mapping of layers but not the set of input files, you can use the Update Mapping button.
You can call the input settings box during a running (but stopped) simulation. The effect of the changed settings is immediate. However, the network is not reinitialized and all other internal structures (e.g. filters, framework) remain the same. This is useful if you want to train a network for different sets of stimuli or use other mapping parameters for parts of a set.

4.2.3 Network settings

The next thing you might want to do is adjusting the parameters for the neural network itself, i.e. which network model should be used (only one is implemented in the current version), how fast it should learn and how strong the influence of the apical dendrite to the cell soma potential should be. Please note that layer sizes are calculated automatically during the initialization of the internal network structures. The desired size of the input layers is used as parameter (be careful with that parameter). In order to understand the internal effects of the parameters you can set please consult [19].
As with the input settings, parameter changes don't require a reinitialization of the network and are valid for all subsequent pictures in the current simulation.

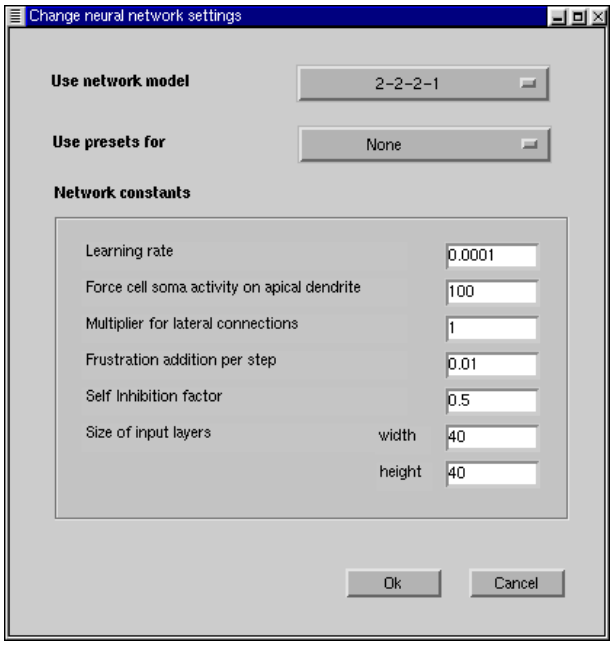


Figure 14: NVS Network settings

4.2.4 Filters

Before the input layers are mapped onto the network, up to three filters are applied serially to the original image. For a description of the different filter types, please see the statistics section. The filter changes are immediate and don't influence other data of the current simulation.

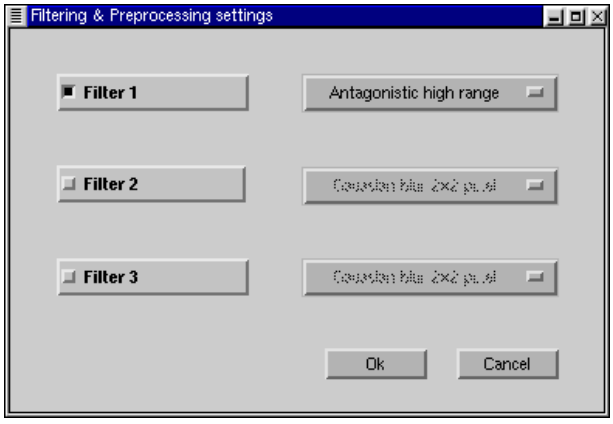


Figure 15: NVS Filter settings

4.2.5 Framework

The settings in this dialog box describe the overall behaviour of the simulation and the graphical user interface. You can select which tasks should be done and if, how often the output of a specific task should be displayed. With the learning option turned off the effect of stimuli on a trained network can be observed. If you run multiple simulations in parallel set the output and intermediate directory to different locations.

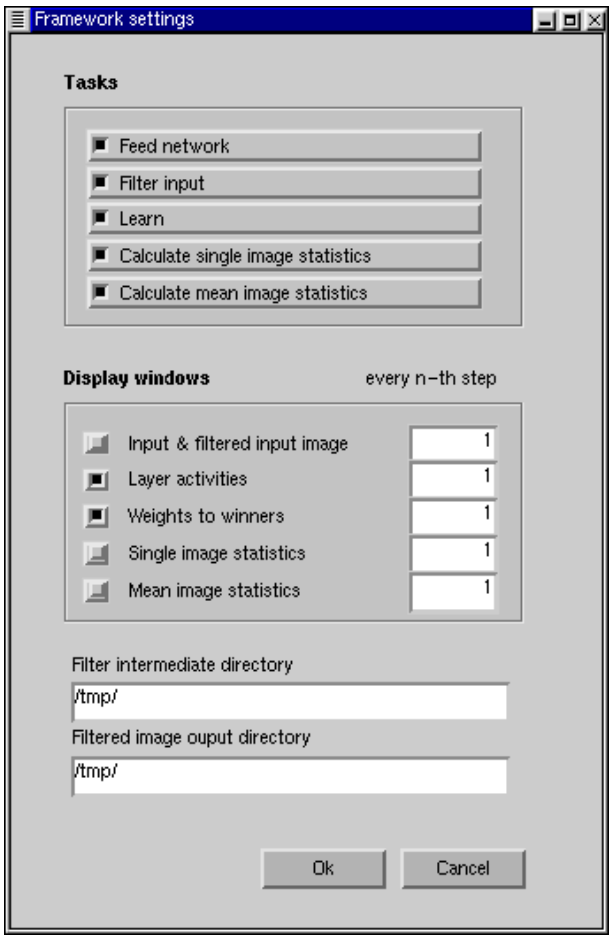


Figure 16: NVS Framework settings

4.2.6 Display menu option

When running a simulation, you can fast-switch on and off the different windows using the display menu. The options correspond to the ones in the framework settings box.

4.2.7 Starting and stopping the simulation

Use the Start button in the control panel to run the simulation. You can interrupt it at any time by clicking once on the stop button. The executing script needs some time to finish the running calculations, so please be patient. While running, the progress is displayed on a progress bar in the lower region of the control panel. When using the same input image for multiple mappings, one step corresponds to one image, not one mapping.

Figure 17: State display while running NVS simulation

Figure 18: Displaying input and filtered input image

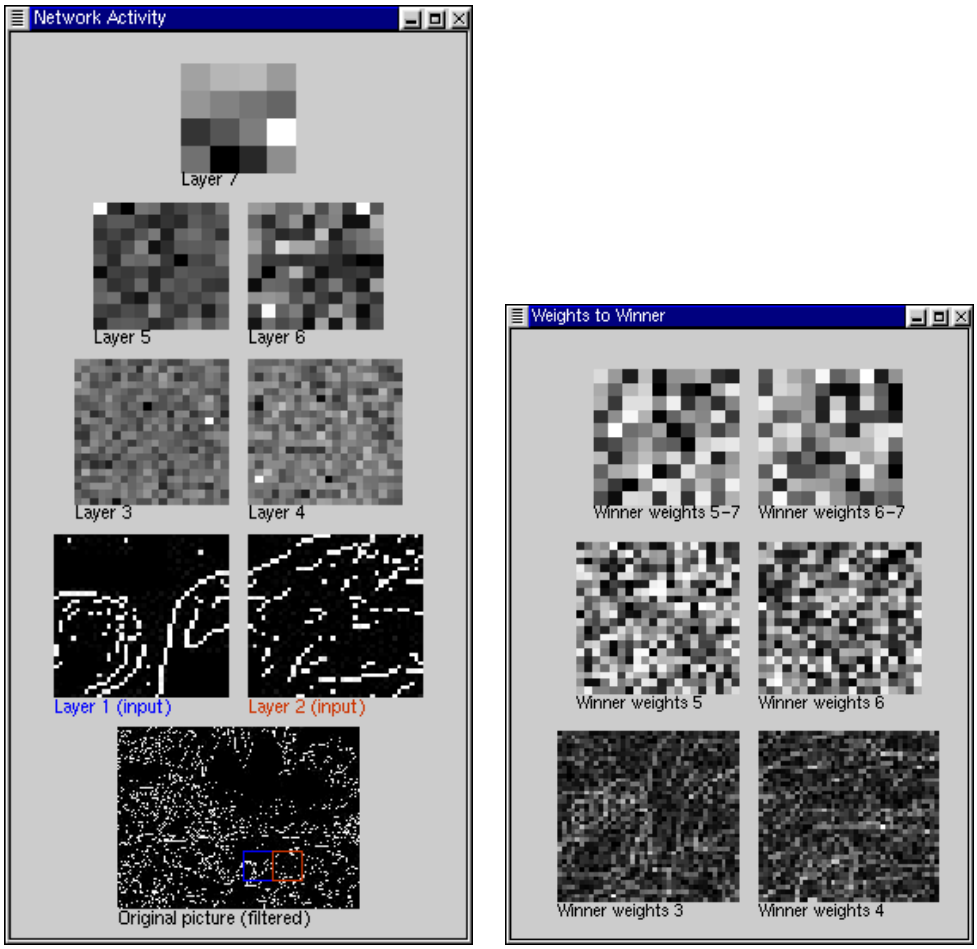


Table 5: Display of network activity and network weights

5 Statistics

The natural visual stimuli collected for our semester work have been statistically evaluated. We found out that natural visual stimuli have quite similar characteristics in the frequency spectrum if we take the mean of a large sequence of images. It also can be shown that the mean intensities of a big set of stimuli produce an image that is greyed and shows no edges anymore. The Marsupilami cartoon that was recorded from TV shows the logo of the TV sender in the mean overall image which of course implies that the upper left corner is not a good natural stimulus.

5.1 Measurements

We calculated the common statistical values for the intensities on the grayed image and on the 2D discrete FFT of this image. This includes mean, variance and standard deviation. The parameters minX and maxX show how strong the values differ over all images in the sequence.

5.2 2D discrete FFT

There are quite some important rules one has to follow when using two dimensional discrete FFT in Matlab. The image has to be padded (with zeros per default) or the transformation will be errenous. The result has to be shifted to obtain the DC coefficient in the middle (as it is common for the continuous FFT). Details about the 2d discrete FFT can be found in [8].
When using zero padding you should not be astonished about high amplitude values on the frequency axes. There is an abrupt intensity change at the image boundaries. We therefore calculated the statistics on the winter image collection a second time after having filtered the intensity image with an ellipsoidal surface. This made the effect on the axis vanish as expected (cf. winterx in Table 7).

5.3 Improving 2D discrete FFT

When using zero padding you should not be astonished about high amplitude values on the frequency axes. There is an abrupt intensity change at the image boundaries. We therefore calculated for winter a second time after having filtered the intensity image with an ellipsoidal surface. This completely vanishes the effect of high amplitudes on the frequency axes.

5.4 Batch statistics using the NVS framework

The NVS framework was enabled to allow calculation of various statistics on a single image and on all images processed so far. The two screenshots

show some sample values and give a good impression of the values provided.

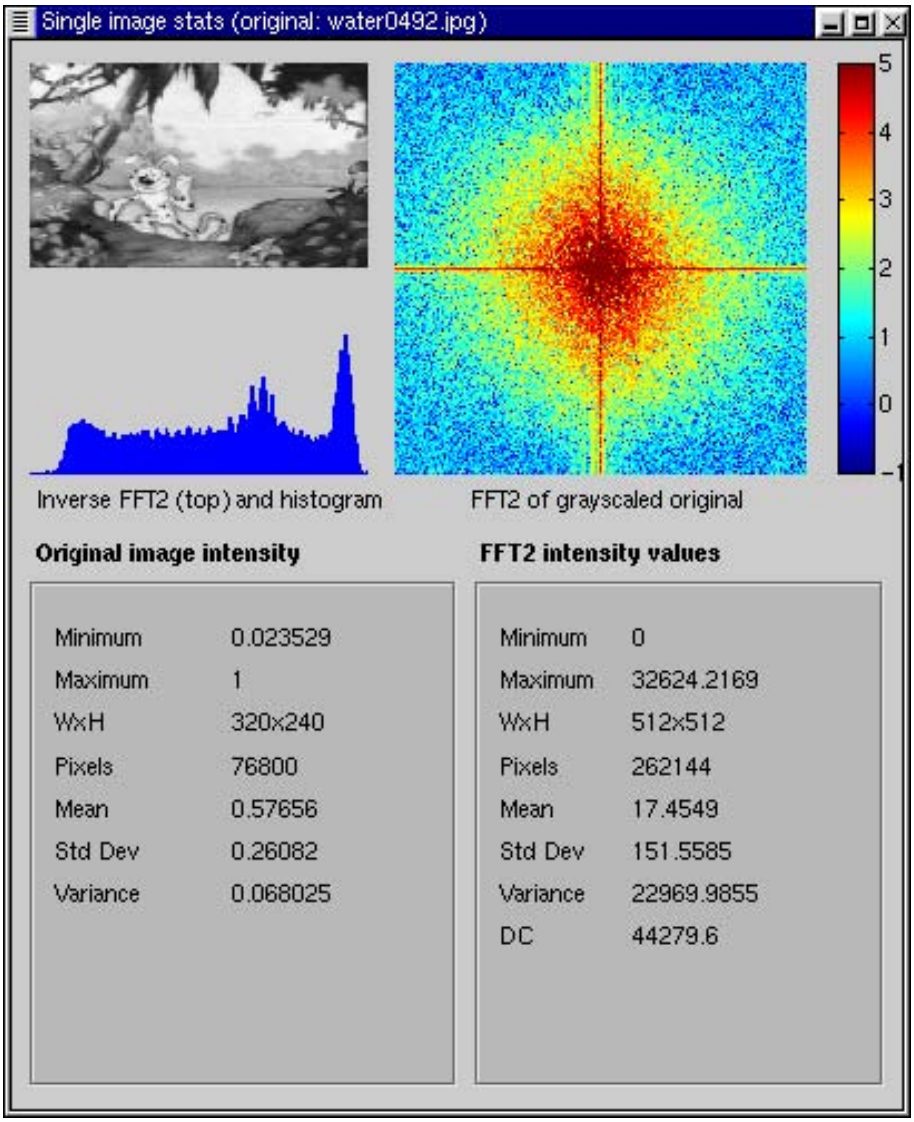


Figure 19: Statistics on a single image

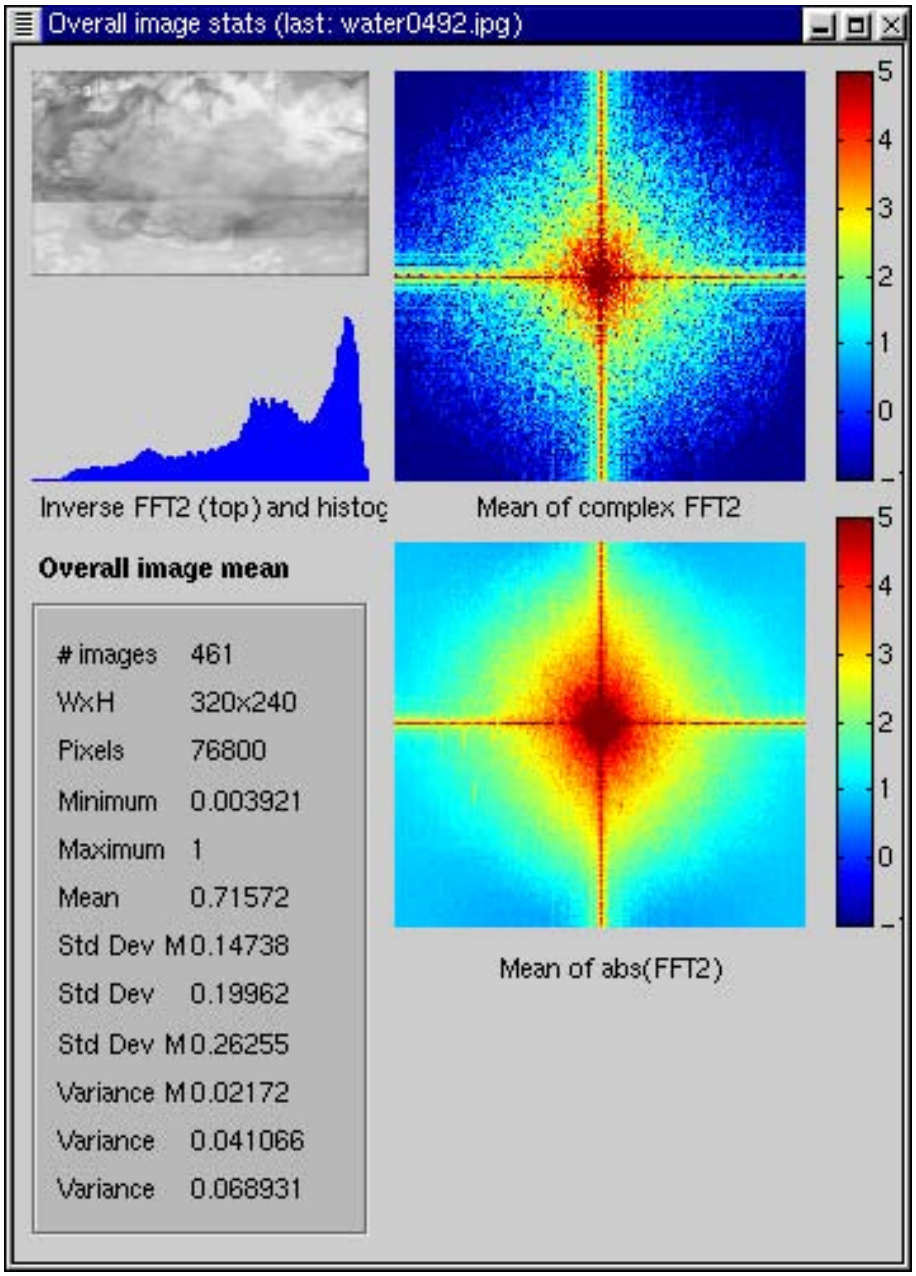


Figure 20: Statistics on a single image

5.5 Results - Plots

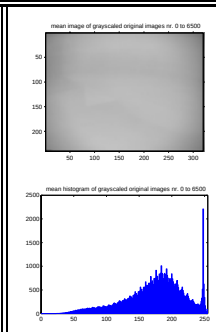
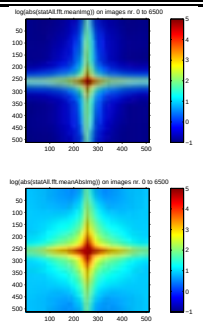
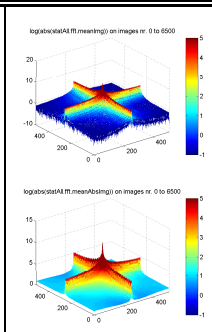
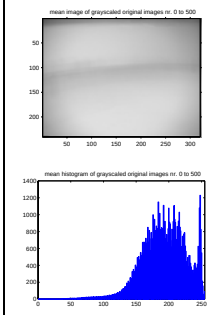
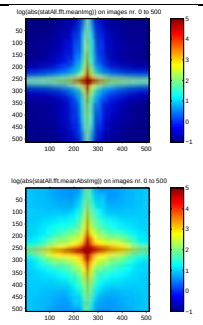
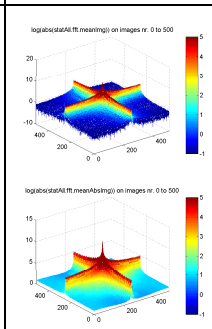
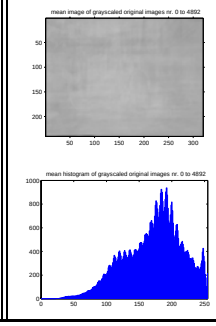
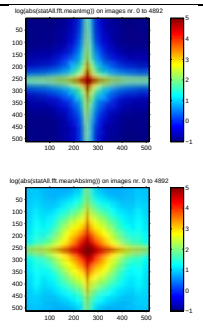
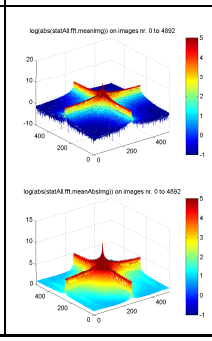
	meanImg	FFTmean2D	FFTmean2D
duplo			
frame			
house			

Table 6: Intensity statistics of the duplo, frame and house image sequences

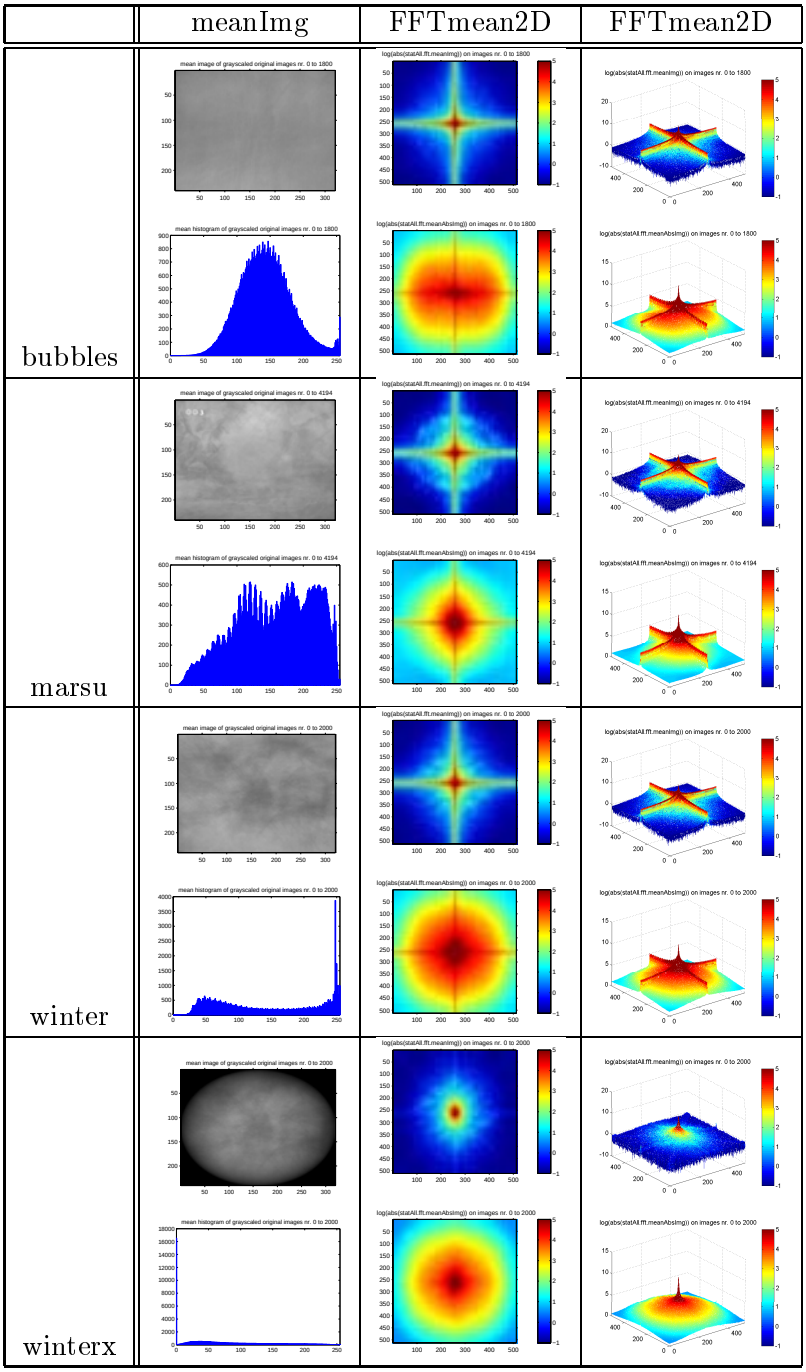


Table 7: Intensity statistics of the bubbles, marsu, winter and winterx image sequences

5.6 Results - Values

Below you find all values calculated on the image sequences using Matlab (and zero padding for 2D discrete FFT). The winterx sequence is the same as the winter sequence except that an ellipsoid was used to filter the intensity image before applying the 2D discrete FFT. You can clearly see the effect of the ellipsoid when looking at the plots in in Table 7. The high amplitude FFT values on the frequency axes have vanished as expected.

	numImg	min	max	mean	meanSd	meanVar
duplo	6501	0.00	1.00	0.68	0.16	0.03
frame	501	0.02	1.00	0.76	0.12	0.01
house	4893	0.05	1.00	0.68	0.15	0.02
intro	125	0.04	1.00	0.15	0.13	0.02
lion	4617	0.00	1.00	0.50	0.34	0.12
marsu	4195	0.00	1.00	0.61	0.21	0.04
bubbles	1801	0.00	1.00	0.00	0.14	0.02
winter	2001	0.00	1.00	0.56	0.30	0.09
winterx	2001	0.00	1.00	0.29	0.26	0.07

Table 8: Statistics from Matlab: `statAll.orig` - part1

	minVar	maxVar	minSd	maxSd
duplo	0.00	0.11	0.01	0.33
frame	0.01	0.02	0.08	0.15
house	0.00	0.09	0.07	0.30
intro	0.00	0.05	0.02	0.21
lion	0.06	0.15	0.24	0.38
marsu	0.01	0.08	0.12	0.29
bubbles	0.01	0.02	0.12	0.15
winter	0.03	0.12	0.16	0.35
winterx	0.02	0.11	0.16	0.33

Table 9: Statistics from Matlab: `statAll.orig` - part2

	numImg	min	max	mean	meanSd	meanVar	meanDC
duplo	6501	0.00	50475.42	0.42	165.37	27538.83	52432.05
frame	501	0.00	46784.26	0.32	178.85	32186.62	58004.41
house	4893	0.00	43182.83	0.60	162.75	26733.93	51898.26
intro	125	0.00	11362.38	0.08	49.83	2762.18	11518.38
lion	4617	0.00	42337.87	0.31	147.76	22254.86	38046.65
marsu	4195	0.00	41849.51	0.55	153.50	23863.73	47028.02
bubbles	1801	0.00	34402.78	0.00	134.57	18318.44	43227.82
winter	2001	0.00	39287.38	0.57	150.79	23006.56	42876.61
winterx	2001	0.00	24791.01	-0.00	96.78	9549.86	22178.88

Table 10: Statistics from Matlab: `statAll.fft` - part1

	minVar	maxVar	minSd	maxSd	minDC	maxDC
duplo	7570.91	51234.27	87.01	226.35	28096.53	74677.77
frame	24576.16	43804.46	156.77	209.30	51196.45	67892.10
house	10408.58	37610.73	102.02	193.93	27551.92	63304.21
intro	687.31	5187.66	26.22	72.03	8579.33	14494.87
lion	9809.81	43022.84	99.04	207.42	24270.38	64952.48
marsu	11840.54	37633.16	108.81	193.99	31748.81	61968.33
bubbles	10398.08	23890.29	101.97	154.56	32156.27	49907.25
winter	7403.18	33349.62	86.04	182.62	23919.11	56732.00
winterx	3248.71	16089.34	57.00	126.84	12685.18	31100.26

Table 11: Statistics from Matlab: `statAll.fft` - part2

6 NVS Implementation

This section describes the different modules of the framework and how they interact with each other and the external world (input/output).

6.1 Module overview

The three primary design goals of the NVS module structure were:

- Small, simple and extensible modules
- Separating data calculation and user interface control
- Well documented reusable modules with a clear interface

Modules correspond to Matlab .m function files. The following drawing shows, which modules call each other. For the UI modules controlPanel, inputSettings, networkSettings, filterSettings and frameworkSettings there exists additionally a callback routine which handles user actions.
For a detailed description of each module, please see below.

6.2 Interfaces

Instead of using Matlab function parameters, modules import structured global variables. The advantage is speed. Global variables are accessed by reference, function parameters by value.
The following global variable structs are used by the NVS modules (for a detailed explanation, see later):

consts	Constants which simplifiy program reading and writing
params	Network parameters like learning rate, negative decay factor and so on.
framework	Everything that has to do with the framework and the current state of the simulation
input	Informations about selected input files and the current input image
filters	Available filters and optimized filter call structure
network	All layers, weights and chosen network model parameters
stat	Saved statistics on current image
statAll	Saved overall statistics

6.3 Description of global variable structs

The purpose of this section is to explain every global structure in detail, so that getting started with the already written code is as easy and smooth as possible. The variables are sorted alphabetically.

*consts.**	
ANDFORWARD = 1	AND-forwarding connection, using $I_j = \sum_i w_{ij} A_i$
MAXFORWARD = 2	Maxmimum forwarding connection, using $I_j = max(\sum_i w_{ij} A_i)$
APICALPROJECTION = 3	Lateral connection on apical dendrite
Used by: initParameters, initNetwork, feedForward, learn, UI/networkSettings.m	

*filters.**	
available	cell array of available filters (id:char, name:char, cmd:char). cmd is the command executed when filtering an image. For examples, see initParameters.
seq	cell array containing names of filters to use (in this order). This information is used primarily by the GUI.
calls	cell array containing Matlab function calls for each selected filter. %i, %o and %t are placeholders for the input and output filename and the temporary intermediate path (all paths are absolute).
Used by: initParameters, initFilters, filterImage, UI/filterSettings	

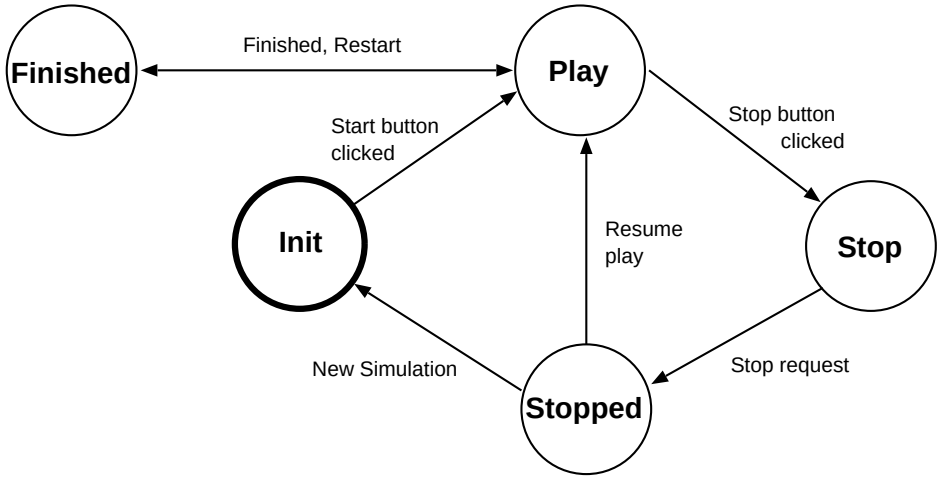


Figure 21: Possible simulation states

*framework.**	
state	Simulation state (see state diagram above), described by the following constants:
IDLE	do nothing...
PLAY	in mainLoop, working
REQUESTSTOP	User requested stop (pressed Stop button)
STOPPED	Out of mainLoop
INIT	Network not yet initialized
stateString	Literal description of simulation state
gcf	Backup of figure number that is being drawn into. When the controlPanel callback function interrupts a drawing process, it restores the gcf to framework.gcf when finished. If this is not done, the value of gcf might be wrong.
startTime	first image index read
endTime	read until this image index
time	current image index
tasks	What the simulation should do (when value is 1). Possible tasks are doFeed, doFilter doLearn, doSingleStats, doMeanStats
display	Which windows the simulation should display and update (when value is 1). Possible windows are images, weights, singleStats, activity, meanStats
drawActivityInterval, drawWeightsInterval, drawInputInterval, drawSingleStatsInterval, drawMeanStatsInterval	How often the separate windows should be updated (if displayed). Value in time steps.
hControlPanel, hControlButton	Handles to control panel figure and control button
menuSim, menuSettings, menuDisplay	Handles to control panel menu bar entries
hState	Handle to control panel simulation state description
pbar	Progress bar object; used by the progressBar module
Used by: initParameters, initFilters, filterImage, UI/filterSettings	

*input.**	
imWidth	Width of input image
imHeight	Height of input image
imType	Type of current input image ('jpg' or 'tif')
desc	Description of input fileset – text, author, date, title
fileFilter	Cell array with file/path selections. This is used by initInput to read in a list of files to be processed.
files	File list structure read in by initInput. Each entry contains a name (filename), pid (path id – index into filePaths array) and a type ('jpg' or tif')
maxOverlap	maximum overlap when mapping input layers (0..1)
posStrategy	chosen mapping strategy, only used by chooseInput and UI
samplesPerImage	How many times to map input layers from the same input image
tempPath, outPath	path for temporary file (filter intermediate) and fully filtered images
currentPictureIndex	index into files array, image currently being processed
currentImage, filtered-Image	planned, but not yet implemented: cache for original and filtered input image
from, to	use input selection [from,to]
isMat	= 1, when a desc.mat file was opened
filePaths	paths of fileFilter selections
viewPositions	current mapping of input layers, set by chooseImage. Matrix containing [left top width height;] for each input layer of the network
thisSample	
filteredPictureName	absolute filename of fully filtered picture
Used by: almost all modules	

*network.**	
modelName	what network model is used (see initNetwork for the description of network models)
numLayers	total number of layers in current model
numInputs	number of input layers in current model
Used by: initParameters, initInput, initNetwork, chooseImage, UI/networkSettings	

*layers{1..numLayers}.**	
settings	size = number of neurons in this layers isInput = 1, if the layer is an input layer
input	vector; size is number of neurons: summed input (is reset at every time step by feedForward)
activity	vector; size is number of neurons: cell some activity of each individual neuron
apical	activity at apical dendrite of each neuron
apical0	saved activity, used by feedForward and learning modules
average	vector; size is number of neurons: average activity of each neuron
apicalaverage	vector; size is number of neurons: average activity at the apical dendrite of each neuron
time	frustration level for each neuron; the higher this value, the more probable a learning event is
winner	number of winner neuron in this layer and time step
connections	cell array, one entry for each connection (layer to layer) targetLayer: which layer the connection is directed at weights: matrix with n*m entries (full connect) type: type of connection (see consts.*) learnrate: learning rate for this layer
numLayers	total number of layers in current model
numInputs	number of input layers in current model
Used by: initParameters, initInput, initNetwork, chooseImage, UI/networkSettings	

*stat.**	
from	statistics beginning at this index
to	statistics calculation ending at this index
singleFFT	2d discrete fft on a single image
sumFFT	summed 2d discrete ffts
Used by: initParameters, mainLoop, calcStatistics, statistic, UI/showSingleStatistics, UI/showOverallStatistic	

statAll.orig – Textual statistics on original image	
curImg	currently processed image
numImg	number of images used for the statistic
sumImg	sum of all image intensity values
sumHisto	sum of all image intensity histograms
sumSd	sum of standard deviation
sumVar	sum of variances
min, max	minimum, maximum of intensity values
minVar, maxVar	minimum, maximum variance
minSd, maxSd	minimum, maximum standard deviation
h, w	height, width in pixel
pixel, sumPixel	pixels per image, pixels of all images
meanImg	mean intensity image
mean	mean intensity value
meanHisto	mean histogram
meanSd	mean standard deviation
meanVar	mean variance
Used by: initParameters, calcStatistics, UI/showOverallStatistic	

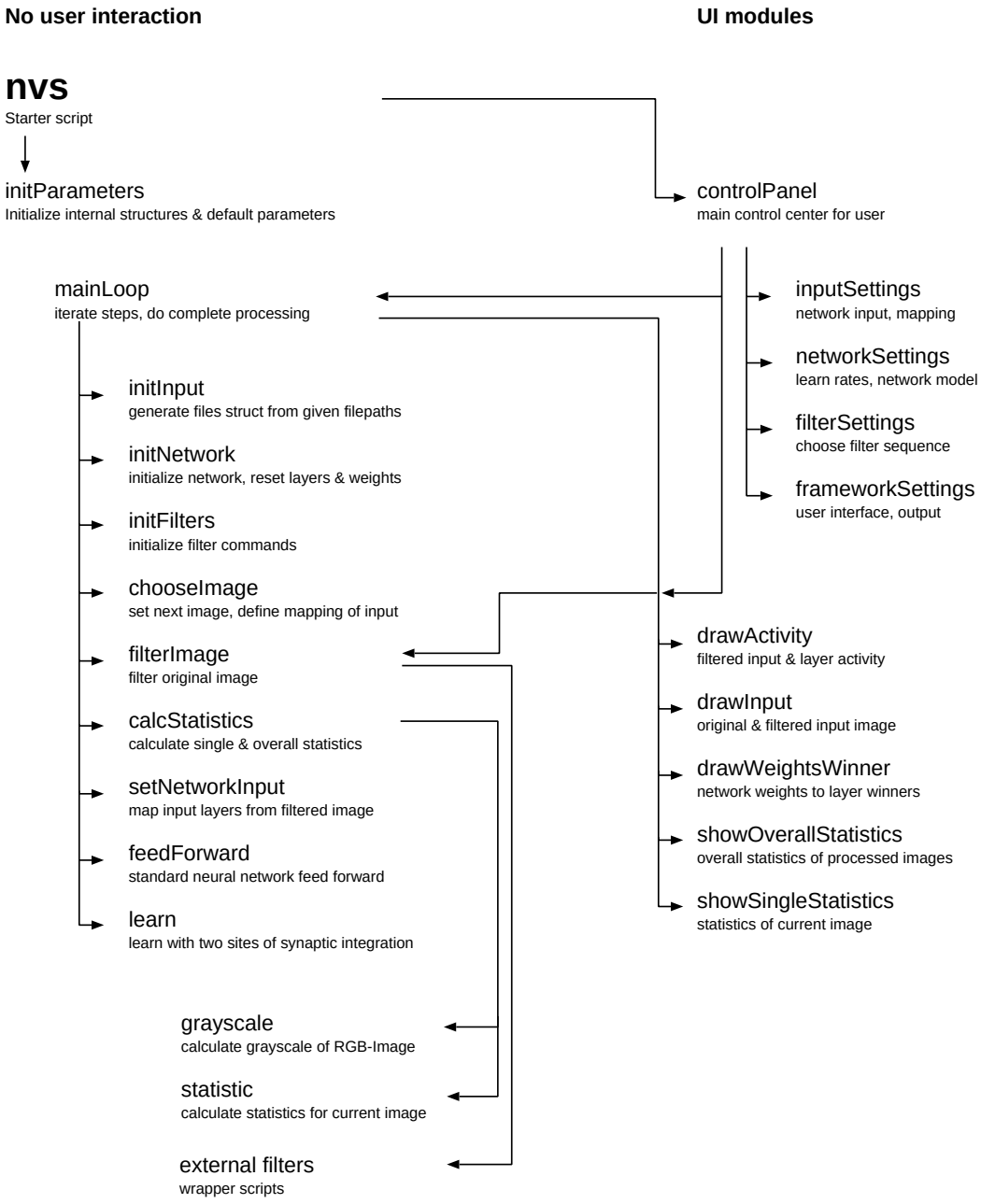


Figure 22: The NVS module structure

6.4 Description of individual modules

This section describes the different modules and their inner working. The code is well documented. For the interaction of modules, refer to the drawing above.

6.4.1 nvs

The `nvs` start script sets additional search paths for Matlab modules (user interface and filters) and calls initParameters to initialize default parameters of the environment. This call is required for the correct working of all other modules. Then, control is transferred to the control panel, which draws the main window and returns afterwards.

6.4.2 initParameters

Initialize all global structures. The following tasks are done:

- Define available filters and external calls, set default filter(s)
- Set default input size, strategy, paths and range
- Reset single image statistics and overall statistics
- Set parameter and constant values
- Random number generator seed initialization
- Default Framework settings (Output options, Tasks)

6.4.3 controlPanel

After the initialization, the nvs module starts the `controlPanel`. This module makes heavy use of graphical user interface elements. It displays a Matlab figure with a background image, a menu bar and a control (Start/Stop) button. If no input has been previously selected, a default text is displayed (and defined here). Another important element in the control panel is the progress bar, which may be initialized here and is displayed by the `progressBar` module.
After finishing drawing, the control panel returns to the calling nvs module, which in term returns control to the Matlab prompt. Control of the simulation is handled by the controlPanel_Callback module.

6.4.4 controlPanel_Callback

When the user requests interaction in the control panel, this function is called. The calling interface element passes a parameter to the callback, which can be as follows:

- `setInput, setNetwork, setFilters, setFramework, about` call the corresponding display window function
- `updateState` redraws the literal simulation state displayed above the progress bar

- `PlayStop`
  If no simulation is running, the framework state changes to PLAY. Before running `mainLoop`, the network is initialized (`initNetwork`) when running the first time. If a simulation is currently running, then the callback has interrupted the mainLoop. To avoid recursion, the callback sets the framework state to REQUESTSTOP. At the end of each time step this variable is checked and the mainLoop can exit. At this point, control goes back to the `controlPanel_Callback` where the mainLoop was called initially. The reason for the polling of the stop condition lies in the asynchronous behaviour of calls to GUI callback functions.

- `dispActivity, dispWeights, dispInput, dispMeanStats, dispSingleStats` are menu callbacks requesting to switch display of windows other than the control panel on or off. The `framework.display variable` and the menu are updated.

- `simExit` Close all windows, clear all variables and exit to Matlab.

- `simNew` Create new simulation; clear all variables and close all windows. Ask user if sure, before doing it.

- `simSaveAs` Save the currently running simulation and settings to a Matlab .mat-File. This may use up *a lot of* disk space.

- `simSaveSettingsAs` Save only the current settings

- `simOpenSettings` Open previously saved settings from Matlab .mat-File

- `simOpen` Open previously saved simulation and settings

On each callback, the internal function `restoreFigure` is called. This is because the callback might interrupt a current draw operation and change the current figure. Other window drawing routines might draw into the control panel accidentally without using this call.

6.4.5 mainLoop

The `mainLoop` is the main iteration loop of the simulation. It gets input data from external sources, applies filters, feeds and trains the network and calculates statistics. It also calls the necessary functions to output and update input, network activity, weights and statistics.
The functionality is encapsulated into small modules (see module overview) which are later explained in more detail and get called in the following context and order:

- `initFilters` before beginning with the loop – to make sure all filter calls are correct and up-to-date.
- `chooseImage` select next input image and map network input layers
- `filterImage` filter image using filter call sequence
- `drawInput` show filtered and unfiltered input image
- `calcStatistics` calculate single image statistics and update overall statistics
- `showSingleStatistics, showOverallStatistics`
- `setNetworkInput` activates the input layer with the chosen mapping
- `feedForward` Activation is passed on to all other layers in the network
- `learn` Update network weights using encapsulated learning rule
- `drawActivity` shows the new layer activities
- `drawWeightsWinner` shows the network weights leading to the layer winner neuron
- `progressBar` update coloured progress bar in control panel

`mainLoop` checks the `framework.tasks` and `framework.display` and only executes functions and display if necessary.
If the user requests a stop (framework.REQUESTSTOP), the mainLoop exits even if not finished yet with the loop. The simulation can later continue at this point.

6.4.6 initInput

This function takes the `input.fileFilter` cell array which is normally set by the inputSettings dialog and gets the corresponding files.
For each seperate path (entry in input.fileFilter array), the file type is detected by looking at the extension of the first file. The files, type and index of path are then appended to the existing `input.files` array, which contains all file information from all paths in the end.
The Matlab `deal` function is used to assign path index and file types for every array element. Unfortunately, this function seems to be quite slow if there are a lot of files.

6.4.7 initNetwork

Before running the simulation, the network parameters, layers and weights have to be initialized. This process depends on the selected network model. Several steps are taken:

1. Set number of layers and layer sizes. The general layer size may depend on the size of the input layers. Note that this size should be a square, so that roots are integer values. This limitation could be removed in a later version by rewriting the `drawWeightsWinner` module.

2. Define connection matrix and type of connections. The size of this matrix is $numLayers^2$ and a connection is created by setting a connection type value greater than 0 for the index (from layer, to layer) in this matrix.

3. The learn rates for each layer are calculated using the global learning rate factor and the layer size.

4. For each layer the input, activity, apical dendrite potential, frustration time, average activity, average apical dendrite potential and the winner neuron are reset.

5. For every connection the connection structure is setup and random weights are assigned.

Now, the network is ready to go!

6.4.8 initFilters

Create the `filters.calls` array containing all Matlab functions to do the whole filtering. The `filters.seq` array containing the filter id's to use is used as source.

6.4.9 chooseImage

This module is split in two parts. First, the next input image for the simulation is chosen, and then the view positions which correspond to the different input layers are calculated, using `input.posStrategy`.
The next input image is found by simply incrementing the index of the files array. A modular wrap-around is used to ensure that there is always a next image (e.g. if the simulation is longer than the image count).
For the view positions, a random strategy is implemented. A square region within the image is randomly chosen, then a second square region is positioned left or right of the first. Value `input.maxOverlap` is the maximum overlap factor that can occur.

6.4.10 filterImage

The current image is fed through one ore more filters. The processing is defined by `filters.calls`. These calls are simple Matlab functions. However, before execution using `eval`, a string search and replace is done for %i, %o and %t, to pass the input filename, output filename and a temporary directory to the script. It's also possible to call external programs, by using the ! Matlab command.
The first filter uses the original image as input. Subsequent filters take its output (which is saved in the intermediate temp directory).

6.4.11 calcStatistics

Read in the original image and convert to grayscale (`grayscale`). Statistics are calculated on this single image using `statistic`). Further, a histogram of intensity values is calculated.
Using the single image statistics, the overall statistics for `statAll` is updated.

6.4.12 setNetworkInput

Read the filtered image into a matrix. For every input layer, the views from `input.viewPositions` have to be mapped onto the network. This includes a matrix to vector conversion, because layer activities are saved in vectors. Also, the scale is adjusted from the range 0..255 (grayscale values) to 0..1 (activity values).

6.4.13 feedForward

There are several steps involved in activating the whole network, i.e. all layers.

1. For all layers in the network, current cell soma activity is passed on to the input in a target layer specified by the connection. Connections from different layers are summed up at the target layer before activation. When forwarding signals, two different types of connections are implemented:

   (a) AND-Forwarding. This is the standard feedforward mechanism in artificial neural networks (summed up weighted activity), using the formula

   $$I_j = \sum_i w_{ij} A_i \tag{6.1}$$

   Implementation of AND-Forwarding is straightforward (vector multiplication).

(b) MAX-Forwarding – Maximum of summed up weighted activity, using the formula

$$I_j = max(\sum_i w_{ij} A_i) \tag{6.2}$$

When implementing maximum feedforward, all possible weights are multiplied with the activity of each neuron in the previous layer. Then the maximum of each column is the value for the neuron in the target layer.

2. Now that all input is summed up, the layer is activated. The summed up input is first smoothed by looking at the mean average activity. Then, the layer self inhibition, which is `params.SELFINHIBITION` multiplied with the mean smoothed layer activity, is subtracted. Negative activity is zeroed. Finally, activation is normalized between 0 and 1. For each layer, the average is updated with current activity.

3. The new activity is fed through lateral connections to the apical dendrite of target layers. There, the potential is summed up again and will define learning impact for each neuron in the target layer.

4. For every layer and neuron, the cell soma activation multiplied by `params.FORCEACTIVITY` is added to the apical dendrite potential and defines the total potential at the apical dendrite.

5. Average potential at apical dendrite is update by a non-linear function.

6.4.14 learn

When all cell activity and potentials at the apical dendrite are calculated, the learning algorithm trains each layer.

1. For each layer find the neuron with maximal potential at the apical dendrite (winner) and set frustation time to zero for this neuron, increase frustation time for all other neurons.

2. Decrease weights by a negative value (`params.NEGATIVEDECAY`)

3. Boost weights of winner neuron (learning event)

4. Add average time to all other weights leading to winner (multiplied with `params.AVGADD`). The probability for a learning event increases with time (frustration).

6.4.15 grayscale

Convert RGB image matrix to grayscale, using `rgb2hsv`.

6.4.16 statistic

Calculation of actual image statistics. For a description of the used methods and algorithms, please see chapter 5 on statistics calculation.

6.4.17 inputSettings, networkSettings, filterSettings, frameworkSettings

For documentation see the source code comments. For usage description, see chapter 4.

6.4.18 drawActivity, drawInput, drawWeightsWinner, showOverallStatistics, showSingleStatistics

For documentation see the source code comments. For usage description, see chapter 4.

A Tools

A.1 nameplus.pl

```
#!/usr/bin/perl
# Batch renaming of files with sequence numbers
# You have to uncomment the mv command to make it work

if (!(@ARGV eq 6)) {
    print "usage:\n";
    print "nameplus <suffix> <oldPrefix> <sizeOldSeq> ";
    print "<newPrefix> <sizeNewSeq> <seqIncr>\n";
    print "e.g. rename intro000.tif - intro876.tif to ";
    print "frame0001.tif - frame0877.tif:\n";
    print " > nameplus tif intro 3 frame 4 1\n\n";
    exit;
}

$suffix         = @ARGV[0];
$old_prefix     = @ARGV[1];
$size_old_seqnr = @ARGV[2];
$new_prefix     = @ARGV[3];
$size_new_seqnr = @ARGV[4];
$increment      = @ARGV[5];

if ($increment > 0) {
    $out = `ls -r *.$suffix`; # reversed
} else {
    $out = `ls *.$suffix`;
}
@name = split('\n',$out);

$templ = "%0" . $size_new_seqnr . "d";

foreach $line (@name)
{
  $num = substr $line,(length $old_prefix),$size_old_seqnr;
  $newnum = sprintf($templ,($increment + $num));
  $newname = $new_prefix . $newnum . "." . $suffix;
  print "rename: $line  --> $newname\n";
## uncomment if you are sure what this script does
#  $do = `mv -f $line $newname`;
}
```

A.2 batch.pl

```
#!/usr/bin/perl
# Batch processing of a customized command
# on all files of a directory

if (!(@ARGV eq 4)) {
    print "usage:\n";
    print "batch <command> <batchpath> <suffix> <arg3>\n";
    exit;
}

$command    = @ARGV[0];
$batchpath  = @ARGV[1];
$suffix     = @ARGV[2];
$arg3       = @ARGV[3];

$out = `ls $batchpath`; # get directory contents
@name = split('\n',$out);

foreach $line (@name)
{
 $cmd = $command . " " . "$batchpath/$line" . " " . \
        "${batchpath}_out/$line" . $suffix . " " . $arg3;
 print "calling: $cmd\n";
## uncomment if you are sure what this script does
  $do = `$cmd`;
}
```

B Bibliography

References

B.1 Image Processing - Mathematics

[1] Luc Van Gool, **Introduction to Image Processing and Analysis**, Script of a lecture hold in winter 1999/2000 at ETH Zurich, Department of Electronic Engineers
Theoretical background on image processing

[2] Bernice E. Rogowitz, Thrasyvoulos N. Pappas (Chairs/Editors), **Human Vision and Electronic Imaging IV**, Proceedings of spie - The International Society for Optical Engineering, Volume 3644, Bellingham, Washington
Overview of ongoing research activities on processing of natural visual stimuli.

[3] Mike Heath, Sudeep Sarkar, Thomas Sanocki, and Kevin Bowyer, **Comparison of Edge Detectors - A Methodology and Initial Study**, Computer Science & Engineering Department of Psychology, University of South Florida, Tampa, Florida 33620, Dec 3, 1996
Comparison of Canny, Nalwa-Binford, Sarkar-Boyer and Sobel edge detection algorithms with an evaluation using human ratings. Canny seemed to be superior to the others.

[4] J. Canny, **A computational approach to edge detection**, IEEE Trans. Pattern Anal. Mach. Intell. PAMI-8, 1986, 679-714 *Canny edge detection algorithm*

[5] S. Sarkar and K. L. Boyer, **Optimal infinte impulse response zero crossing based edge detectors**, CVGIP: Image Understanding 54, 1991, 224-243 *edge detection algorithms*

[6] Markku Hauta-Kasari, **Computational techniques for spectral image analysis**, Lappenranta University of Technology, 1999, 1st part
Theoretical background about image processing

[7] **http://www.mathworks.com/**
The manufacturer of Matlab provides a searchable support database, user contributed m-files and guides to Matlab

[8] Documentation of the **Image Processing Toolbox**, http://www.mathworks.com/
Powerful tools for image manipulation in Matlab

[9] KUIM Toolkit, **http://www.ittc.ukans.edu/ jgauch/kuim/kuim.html**
Image processing toolkit supplying image filters in C source code using the JPEG library

B.2 Image Processing - Biological Background

[10] David H. Hubel, **Eye, brain and vision**, Scientific American Library cop. 1995 VIII, 242 p., New York [etc.]
Thorough explanations about how the human visual system transforms visual stimuli.

[11] Randolph Blake, **Cat spatial vision**, Elsevier Publications, Cambridge, 0378 - 5912/88, TINS, Vol. 11, No. 2, 1988 *Fascinating details about how good a cat can see.*

[12] K.-P. Hoffmann, C. M. Morronet, and J. H. Reuter, **A comparison of the responses of single cells in the lgn and visual cortex to bar and noise stimuli in the cat**, Abteilung für Vergleichende Neurobiologie, Universität Ulm Oberer Eselsberg, D-7900 Ulm, Federal Republic of Germany
Investigations on behaviour of LGN cells when a stimulus (bar) is surrounded by partly masked other stimuli

[13] G. A. Orban, K.-P. Hoffmann, and J. Duysens, **Velocity Selectivity in the Cat Visual System. I. Responses of LGN Cells to Moving Bar Stimuli: A Comparison With Cortical Areas 17 and 18**, Laboratorium voor Neuro- en Psychofysiologie, Katholieke Universiteit te Leuven, Campus Gasthuisberg, Herestraat, B-3000 Leuven, Belgium
Measurements of velocity characteristics and response latency of LGN cells to moving bar stimuli.

[14] David Ferster and Bharathi Jagdeesh, **Nonlinearity of Spatial Summation in Simple Cells of Areas 17 and 18 of Cat Visual Cortex**, Department of Neurobiology and Physiology, Northwestern University, Evanston, Illinois 60208
Spatial frequency measurements and comparisons of areas 17 and 18 to LGN cells

[15] **http://www.shortcourses.com/book01/07-03.htm**, *Information about the TIFF image file format*

[16] **http://www.shortcourses.com/book01/07-03.htm**, *Information about the JPEG image file format*

[17] **http://www.wlv.ac.uk/ c9653177/avi.html**,
http://web.cs.mun.ca/k12media/resources.formats.video.avi.html
Information about the AVI video file format

B.3 Image Processing - Neural networks

[18] Konrad P. Körding and Peter König **A learning rule for dynamic recruitment and decorrelation**, Institute of Neuroinformatics, ETH/UNI Zurich, Wintherthurerstrasse 190, 8057 Zurich, Switzerland
A learning rule for neural networks based on physiologically inspired results

[19] Konrad P. Körding and Peter König **Learning with two sites of synaptic integration**, Institute of Neuroinformatics, ETH/UNI Zurich, Wintherthurerstrasse 190, 8057 Zurich, Switzerland
An neural learning rule for unsupervised neural networks based on physiologically inspired results

C Acknowledgements

We want to conclude with saying thank you to Dr. Peter König, Konrad Körding and Prof. Bernasconi who supported our interesting work at the Institute of Neuroinformatics INI at the University/ETH Zurich and gave us many useful suggestions and ideas concerning our semester work.