University of Pisa
Master's Degree in Computer Engineering
Intelligent Systems Project

NEURAL COLOUR COMPARISON

FILIPPO SCOTTO
Student ID: **501743**
Academic Year: **2017/18**

Abstract

Colour comparison is nowadays a major task in industrial printing; providing a high quality copy of a master patch is not easy and comparing them is the only way to achieve good quality copies. However, inspection through traditional approaches (e.g. experts or colour difference function) are not very accurate. In this project a new neural network-based approach is proposed.

The project is composed by two parts, in the first it was developed and trained a function fitting neural network which is able to reproduce the behaviour of the ***colour difference function (CIE DE 76)*** defined by the *CIE* to evaluate the difference between two colours in the *L*a*b* space*.
Starting from a dataset of 1269 master patches, a set of perturbated copies was generated (10 copies for each master), finally the target set was obtained by applying the DE function to the LAB representation of each pair. The resulting dataset was used to trained the neural network. The network takes as input a *subset* of the features extracted from the spectra, and it is able to produce the difference with a regression coefficient value of **0.99682**. However, DE function is very problematic and not so accurate in some cases – *dark colours, blue-violet region, and more*.

The second part of the project is meant to get over the limits of the DE function by using a ***fuzzy interference system*** to adjust the behaviour of the neural network. The adjusted values of DE were then used to train once again the neural network from part one. Even though there has been a decline in the performance (regression coefficient value: **0.98118**), this neural system can be considered more reliable for industrial applications.

Contents

Introduction

The problem with colours in printing industries

Printing as an industrial process

Colour is a physiological *sensation* that our brain uses to recognize the objects around us. In order to reach such sensation four steps are required:

1. a light source emits a light;
2. an object modify and reflects it according to its surface characteristics;
3. the human eye detects the reflected eye and turns it into a stimulus;
4. the brain processes this stimulus and generates the chromatic sensation.

Printing is nowadays a pervasive industrial process; manufacturers of coloured products are expected to maintain high levels of colour quality to guarantee customers' satisfaction. However, as we've already said, a colour is a sensation, furthermore it is highly affected by the environment (*not only not all the light sources emit the same wave, but a light source may vary its characteristics over time!*). Not satisfying customers, means a lot of money loss. So, it is kind of a big deal.

Visual inspections, performed by experienced employees, are not very accurate, each worker may have different colour sensitiveness – *e.g. mood, age, focus are all factors affecting the measure*.

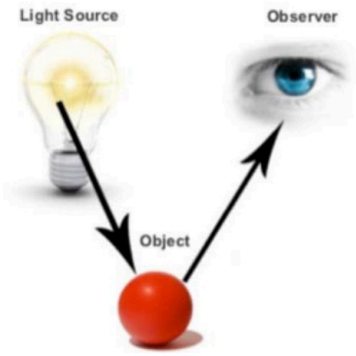


Figure 1.1: Elements of colour perception

CIE L*a*b* Colour Space

The *Commission Internationale de l'Eclairage* (CIE) is the international authority of light, illumination, colour and colour space. In order to measure the difference between two colors, in the 1976 introduced the L*a*b* colour space and a mathematical formula.

The ***L*a*b* colour space*** describes all perceivable colours in three dimensions: lightness, green-red colour opponents, blue-yellow colour opponents. Lab's creators wanted to design a perpetually uniform space, in order to approximate human vision as close as possible. But, as it will be described

in the next chapters, things are a little bit difference. First of all, it should have been represented as a sphere, but it is ***not a perfect sphere*** (*more details and further issues are described in the second part of the project*).

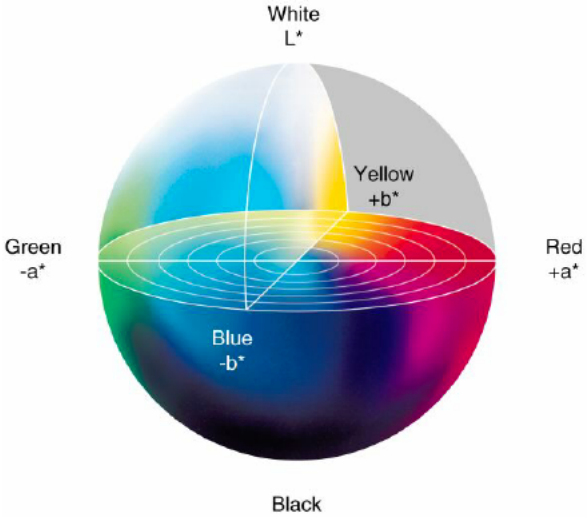


Figure 1.2: The CIE L*a*b* colour space

The distance is then evaluated as an Euclidean distance in the L*a*b* space:

$$\Delta E = \sqrt{(L_2^* - L_1^*)^2 + (a_2^* - a_1^*)^2 + (b_2^* - b_1^*)^2}.$$

- **0 < ΔE < 1**: an observer does not notice the difference;
- **1 ≤ ΔE < 2**: only experienced observers can notice the difference;
- **2 ≤ ΔE < 3.5**: also non-experienced observers can notice the difference;
- **3.5 ≤ ΔE <5**: clear difference between the colours;
- **5 ≤ ΔE**: completely different colours;

Unfortunately, there are regions where an observer perceives a difference between two colors only when they have a ΔE value that is much more higher than 3.5 – *e.g. the dark region of the L*a*b* colour space. Limits of the DE function will be analyzed better in the second part of the project.*

Project outline

The considered scenario in the assignment consisted of a master color patch and an industrial process that prints copies of the master. The more accurate the process the more similiar each copy is to the master. The aim of the project is to define a neural colour comparison not affected by imprecision and subjectivity (*frequent in visual inspections*).

To realize the network, it was provided a dataset containing the master colour patches and the ***optprop MATLAB toolbox***. The project development, here documented, will follow these steps:

1. The *analysis of the dataset* and the *generation of a set of noisy copies* for each master;
2. First part*: Features extraction/selection* and *function fitting neural network definition*;
3. Second part: Development of a *fuzzy inference system* to adjust the differences.

The Dataset

The starting dataset and the generation of the copies

The starting dataset

The dataset consists of **1269** reflectance spectra. Each spectrum is related to a master colour and it is measuerd with 1nm step, ranging from 380nm to 800nm. This results into **421** samples for each spectrum.

The data are provided in a MATLAB file that contains two matrices:

- `spectra[421, 1269]`: each column corresponds to a color patch, and each row to one of the samples;
- `coordinates[6, 1269]`: each column corresponds to a color patch, rows 1 to 3 contain the RGB coordinates; rows 4 to 6 contain the L*a*b* coordinates calculated using the **D65**[1] light source.

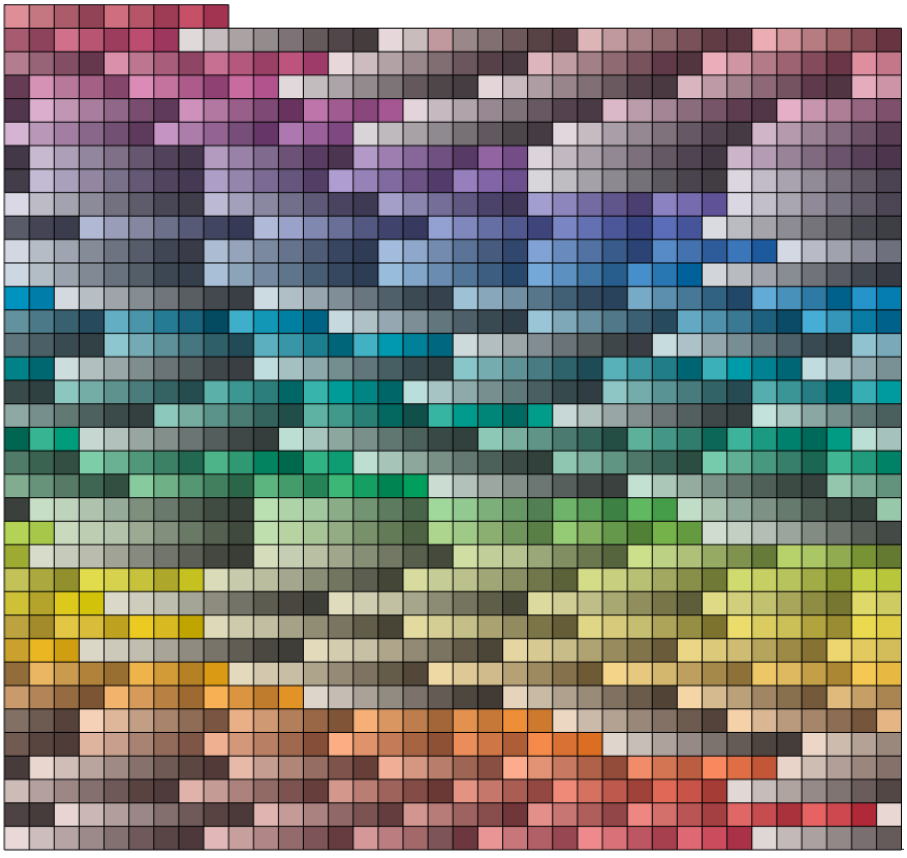


Figure 2.1: Two dimensional visualization of the starting dataset (sRGB)

[1] The CIE has standardized light sources, thus assigning each of them to a standardized *spectral power distribution* (SPD). These standardized SPDs are called *standard illuminants*. D65 is the standard for *daylight*.

Adjusting the dataset

The spectra provided with the starting dataset use a format that is not compatible with the functions defined the optprop toolbox. Since those functions are necessary and will be used several times in the project development, it was necessary to multiply the matrix **spectra**[1269x421] by 100.

It was decided to ignore the **coordinates** matrix and to use instead the two functions **roo2rgb** and **roo2lab** to obtain the RGB and LAB values for each master color.

Copy generation: a naive approach

The dataset as it was provided was not enough to train a neural network, starting from the masters colour patches, it was generated a set of synthetic copies. The copy generation is a *crucial part* of the project. The accuracy of the neural network will depend on how well the generation was.

However, it is well known that keeping things simple is always the best way to start a project. That's why it was decided to start from a naive generation, obtained by simply adding a gaussian noise to the reflectance curves of the master dataset:

```
rng(1); % needed to keep the dataset repeatable
for idx=1:num_copies
  copies(idx).spectra = master*unifrnd(minNoise, maxNoise);
end
```

The first thing that must be decided is the number of copies needed. Heuristically it was decided to have **10 copies** for each master patch. The new dataset will contain **12690 colours**.

Every copy is obtained by multiplying the whole master matrix by the result of the MATLAB function **unifrnd** – *a function that returns an array of random numbers generated from the continuous uniform distributions with lower and upper endpoints specified by minNoise and maxNoise.*
The parameters of the distribution were chosen empirically to obtain a maximum difference of 5 (*circa*) between the master and copy. It was also decided to discard the not-so realistic case of a perfect copy. The distribution endpoints were set as: `minNoise = 1.015` and `maxNoise = 1.135`.

This simple generation has a little problem with the distribution of the distance DE over the whole dataset. As it shown in *figure 2.2*, the distribution is much more denser around zero values, we want to avoid this behaviour that might result in a situation of *over-fitting*.

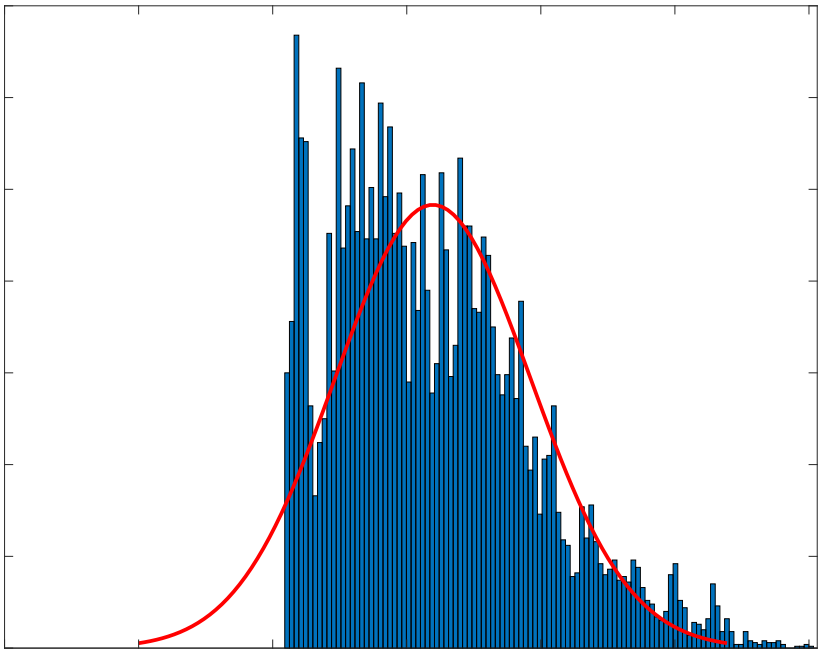


Figure 2.2: Distribution of the difference DE between all the pairs (first algorithm)

Copy generation: a slightly more complex approach

To avoid the undesired behaviour of the DE distribution, it was decided to adopt a different algorithm for the generation of the copies (*num_copies and minNoise and maxNoise did not change*):

```
rng(1); % needed to keep the dataset repeatable
for idx=1:num_copies % 10
  for idy=1:dataset_size % 1269
    copies(idx).spectra(idy, :) = master(idy, :)*unifrnd(min, max);
  end
end
```

This new algorithm produces a much more uniformed distribution for DE (*Figure 2.3*).

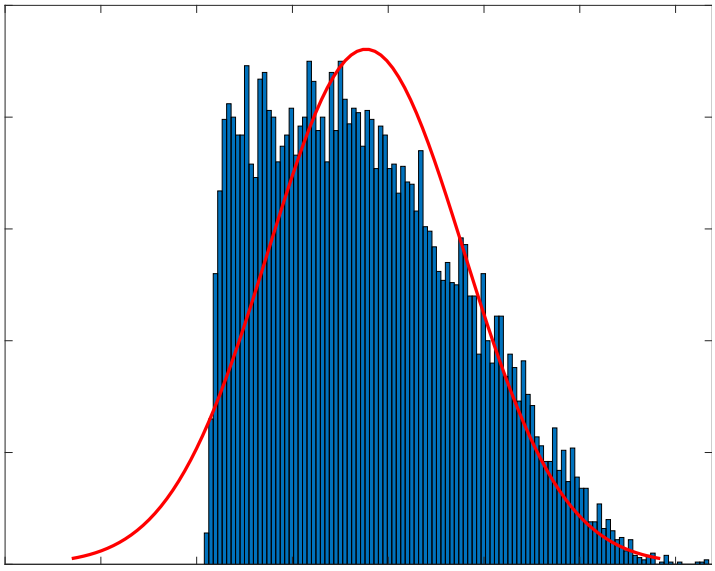


Figure 2.3: Distribution of the difference DE between all the pairs (second algorithm)

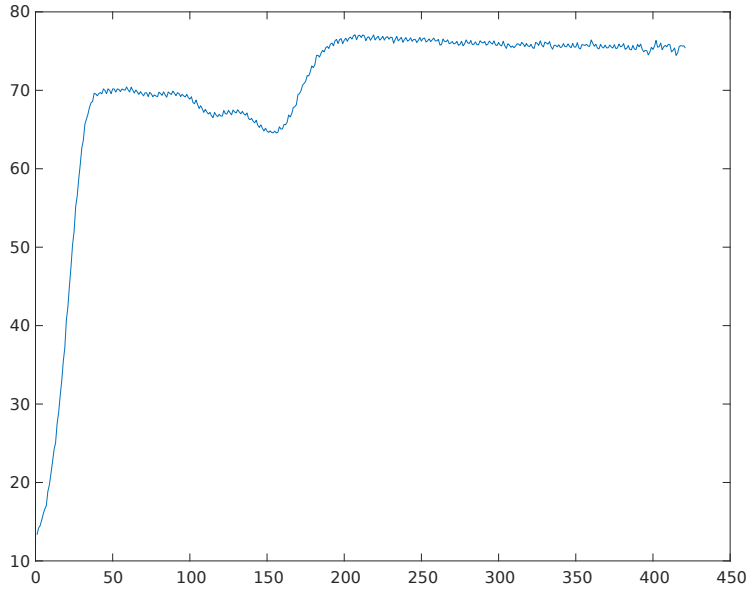


Figure 2.4: Master spectrum corresponding at the first patch in the dataset

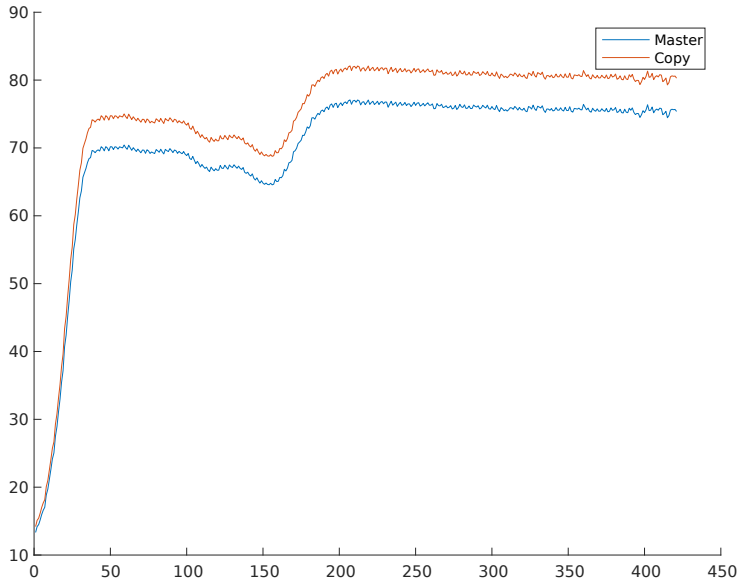


Figure 2.5: Master spectrum compared to its first copy

Figure 2.6: First master patch (darker) compared to its first copy (brighter), DE: 2.2043

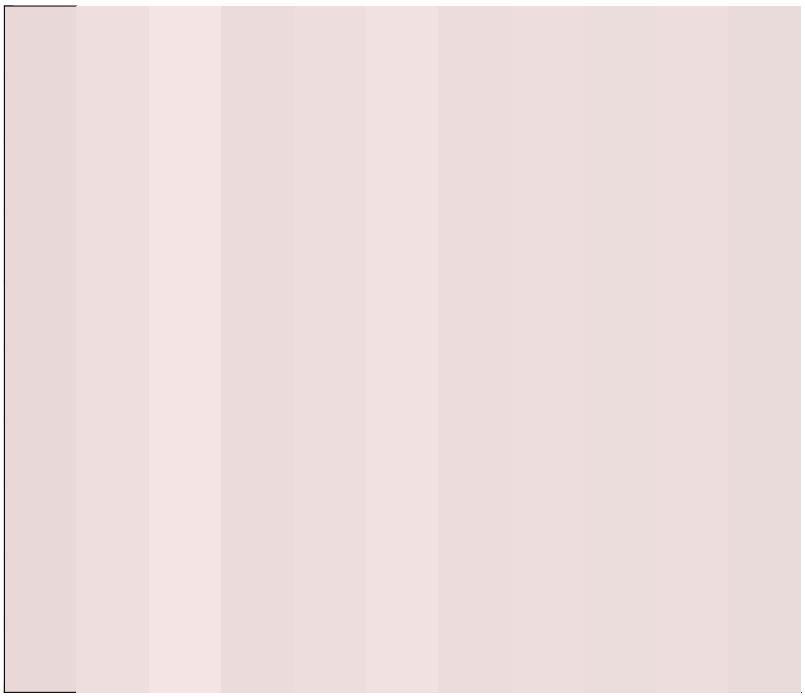


Figure 2.7: First master patch compared to all its ten copies

The new dataset

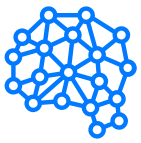
To conclude this chapter of the documentation, let's analyze the adjusted dataset. It is composed by an array of **12690** elements, each of them has the following structure:

MASTER	COPY	DE	MCOLOR	CCOLOR

spectrum[1][421]	spectrum[1][421]	Difference	{
			rgb[1][3],
			lab[1][3]
			}
				{
				rgb[1][3],
				lab[1][3]
				}

Row containing the 421 samples for the master patch.	Row contaning the 421 samples for the copy.	Difference evaluated using the function: **`de(LABM, LABC)`**	Struct containing the RGB and LAB representation of the master.	Struct containing the RGB and LAB representation of the copy.

The columns **`MASTER`**, **`COPY`** and **`DE`** will be used to feed the neural network.



Part 1: The Function Fitting Neural Network

Extraction and selection of the features and neural network design

The aim of this part of the project is to develop a neural network-based system that compares a master and a copy (pairs of spectra) and produces the DE evaluation. For simplicity, in order to find the best representation of colours, we consider the DE as the desired behaviour of the network.

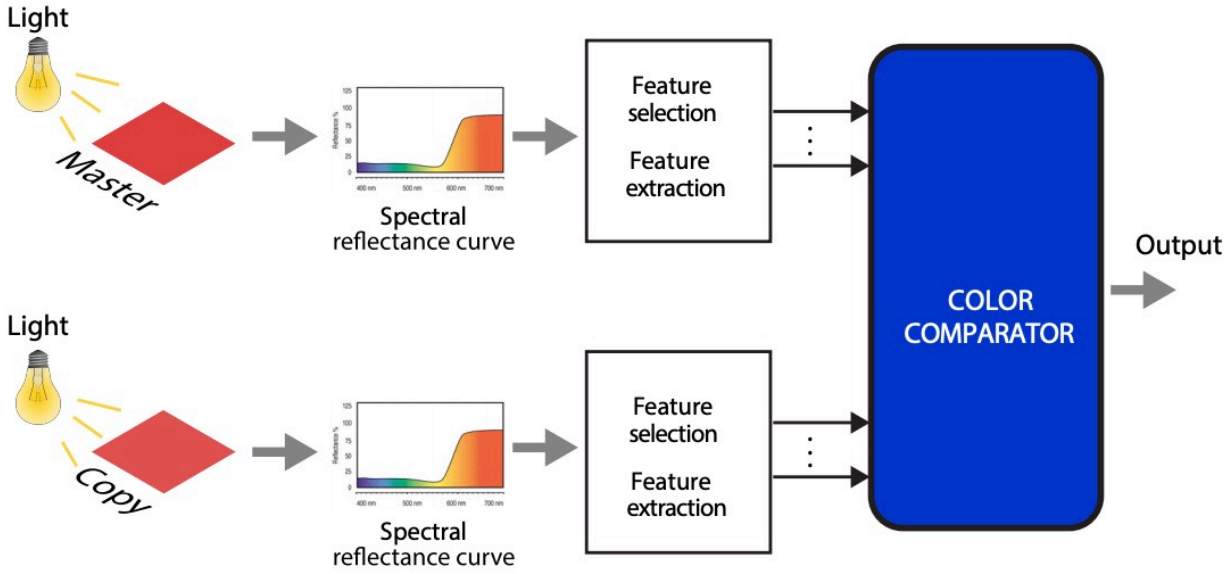


Figure 3.1: The scheme of the neural network developed in the first part of the project

NOTE ON THE EXPERIMENTS
Please notice that all the tests and experiments and the temporal measurements were done on a *iMac Late 2013* with the following hardware configuration:

- **CPU**: 3.2GHz Intel Core i5
- **RAM**: 8 GB 1600 MHz DDR3
- **GPU**: NVIDIA GeForce GT 755M 1024 MB

Features Extraction

The dataset obtained in the previous chapter is definitely too big to be used to feed a neural network – *a matrix with 12690 rows and 842 columns*. It is necessary to reduce the *dimensionality* of the information provided and possibly maintaining the same quality.

The idea is to divide the spectrum in k wavelengths ranges and internally aggregate the information. Once again chosing the right parameter is crucial: k too small might result in information loss; whereas, on the other hand, with a high value of k the risk is the sub-sampling[2].

To find the best solution some experiments are needed.

Here it is the snippet of code[3] used to determine the aggregated version of the spectra (of both copies and masters). The constant **`compress_rate`** is the compression rate and it specifies how many samples should be aggregated. The aggregation was then accomplished by adopting the statistical mean of the **`compress_rate`** samples.

```
for idx=1:12690
  m = dataset(idx).master;
  c = dataset(idx).copy;

  m_compressed = accumarray(ceil((1:numel(m))/compress_rate)',m(:),[],@mean)';
  c_compressed = accumarray(ceil((1:numel(c))/compress_rate)',c(:),[],@mean)';

  netInput(idx) = [m_compressed c_compressed];
end
```

The first attempt was to use a **`compress_rate`** equals to 5, but as it possible to see in *Figure 3.2*, the aggregation appears really dense. The features extracted from each spectrum are **85**, which means a total input matrix of **12690** rows and **170** columns. A neural network fed with this input matrix would be probably very accurate, but it may require too much time!

[2] Sub-sampling is a situation in which the granularity of the information is unnecessarily too high and might slow down the performance of the system.

[3] Please notice that the code was simplified in order to improve the readability of the code!

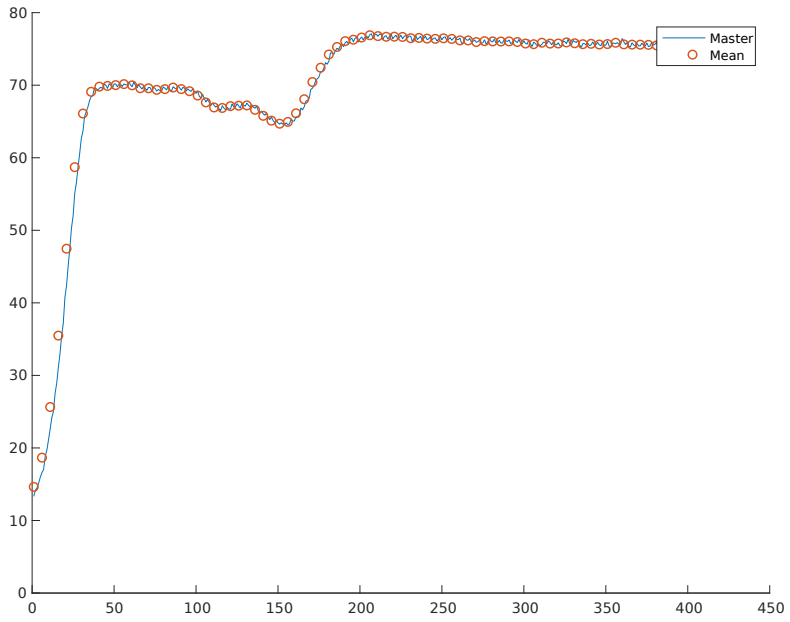


Figure 3.2: First attempt of feature extraction with compression rate equal to 5

The second attempt was done using a **`compress_rate`** set to **10**; in this case the input matrix will be made of **12690** rows and **86** columns. Once again, this is a sub-sampling situation:

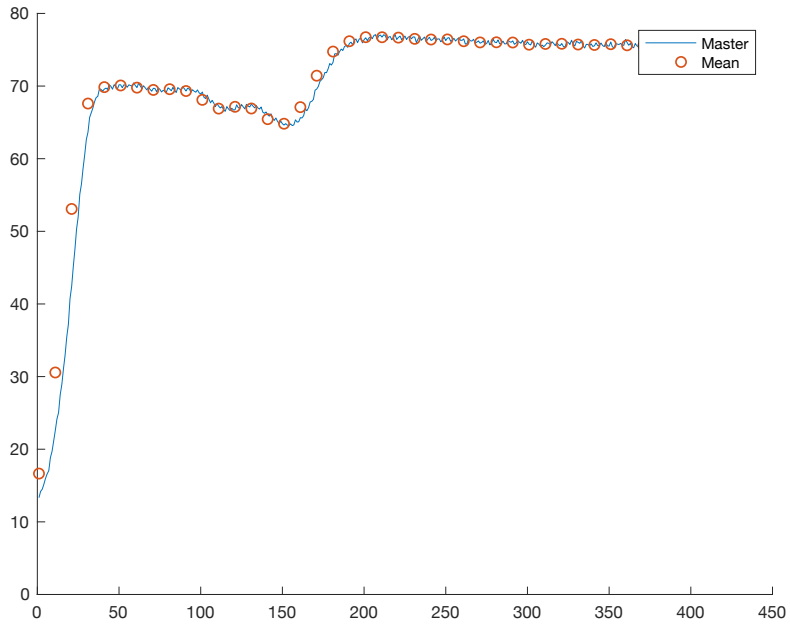


Figure 3.3: Second attempt of feature extraction with compression rate equal to 10

In the third attempt the **`compress_rate`** was set to **20**, resulting in a matrix with **12690** rows and **44** columns:

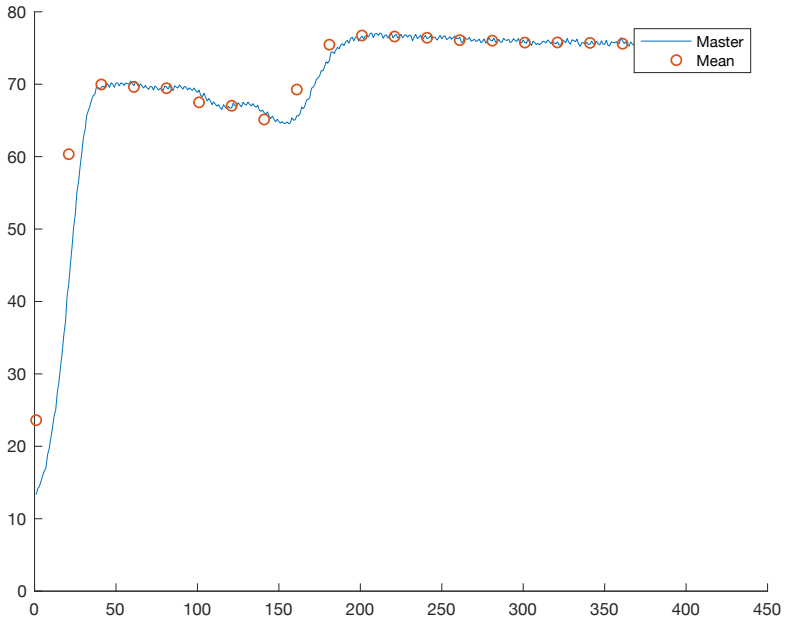


Figure 3.4: Third attempt of feature extraction with compression rate equal to 10

This was probably a good starting point, but it was decided to push further the compression, so in the final attempts it was set a `compress_rate` equals to **40** and **50**, resulting in a matrix respectively with **12690** rows and **22** columns and **12690** rows and **18** columns.

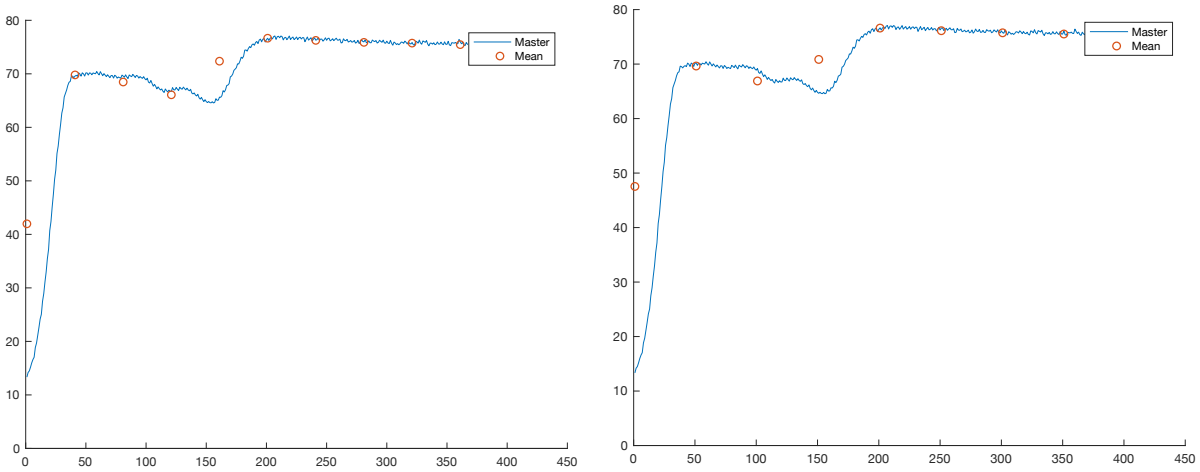


Figure 3.5: Last attempt of feature extraction with compression rate equal to 40 and 50

Developing the neural network: Introduction

This is the core of the first part of the project, the actual development of a neural netowrk capable to reproduce the behaviour of the colour difference function defined in the optprop toolbox (also known as DE).

For this purpose it was decided to use a function fitting neural network with a set of input nodes, one hidden layer and one output node.

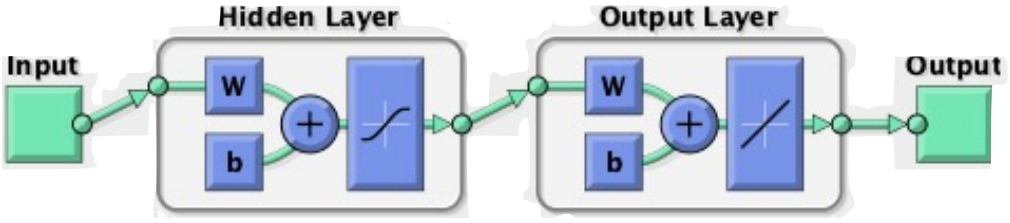


Figure 3.6: An example of function fitting neural network

All the attempts and experiments was done using the following algorithms:

- **Data division**: Random, with *60% training, 20% testing, 20% validation*;
- **Training**: Levenberg-Marquardt;
- **Performance**: Mean Squared Error;

In the beginning it will be considered as input the matrices extracted from the master-copy spectra (*see "Features Extraction" pargraph*), but eventually it will be necessary to perform a *features selection* operation.

Developing the neural network: Attempt #1

It was considered as input the matrix obtained by extracting features with a compression rate equals to 5, which means a matrix **12690x170**.

- **Input Nodes**: 170;
- **Hidden Nodes**: 10;
- **Required time to train**: 27 minutes and 18 seconds;
- **Regression**: 0.99999;

Indeed, results are great (*apart from required time to train*), however, this neural network is way too much complex for this problem. This network was designed only for testing the *MATLAB* environment and the *nntool*.

What is interesting about this first attempt is that if we decrease the number of hidden nodes down to just three we obtain the following results:

- **Input Nodes**: 170;
- **Hidden Nodes**: 3;
- **Required time to train**: 1 minute and 35 seconds;
- **Regression**: 0.9996;

We've lost a little bit in regression, but the required time to train the network is ~*94%* better than the previous one.

Developing the neural network: Attempt #2

Let us consider now the input matrix obtained with the features extraction compression rate set to 10.

HIDDEN NODES	TIME TO TRAIN (mm:ss)	REGRESSION VALUE
5	01:11	0.99994
3	00:06	0.99958

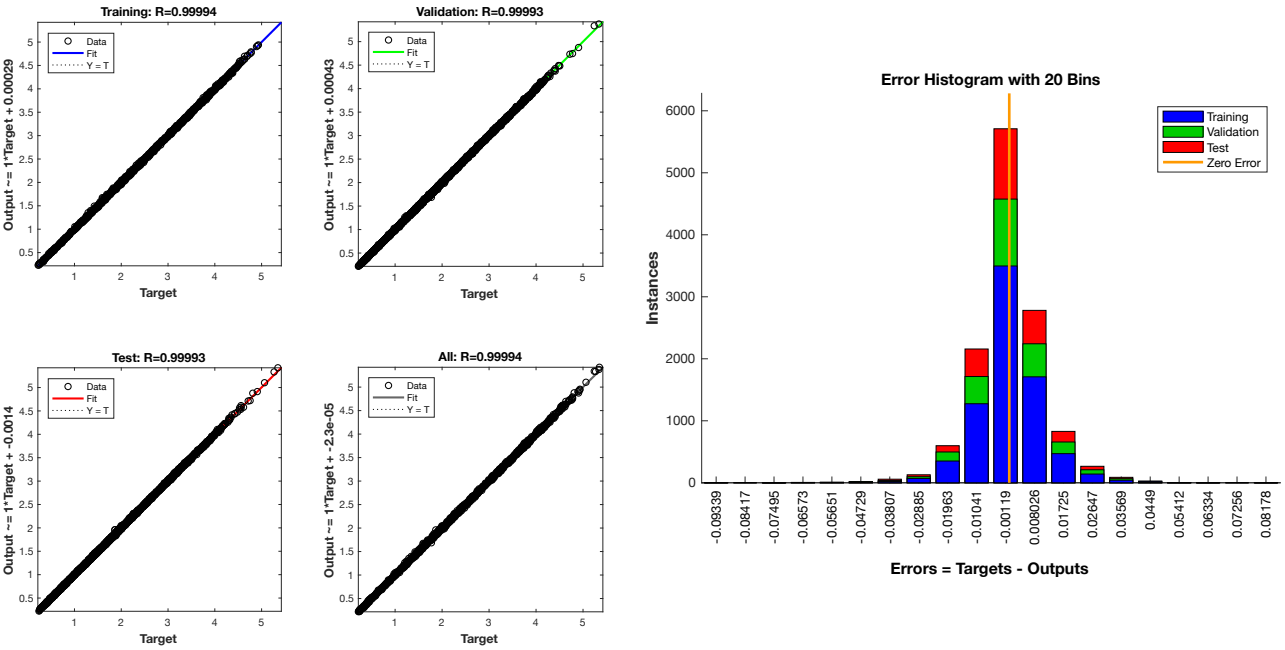


Figure 3.7: Regression plot and Error Histogram of a NN, CR=10 and HN=5

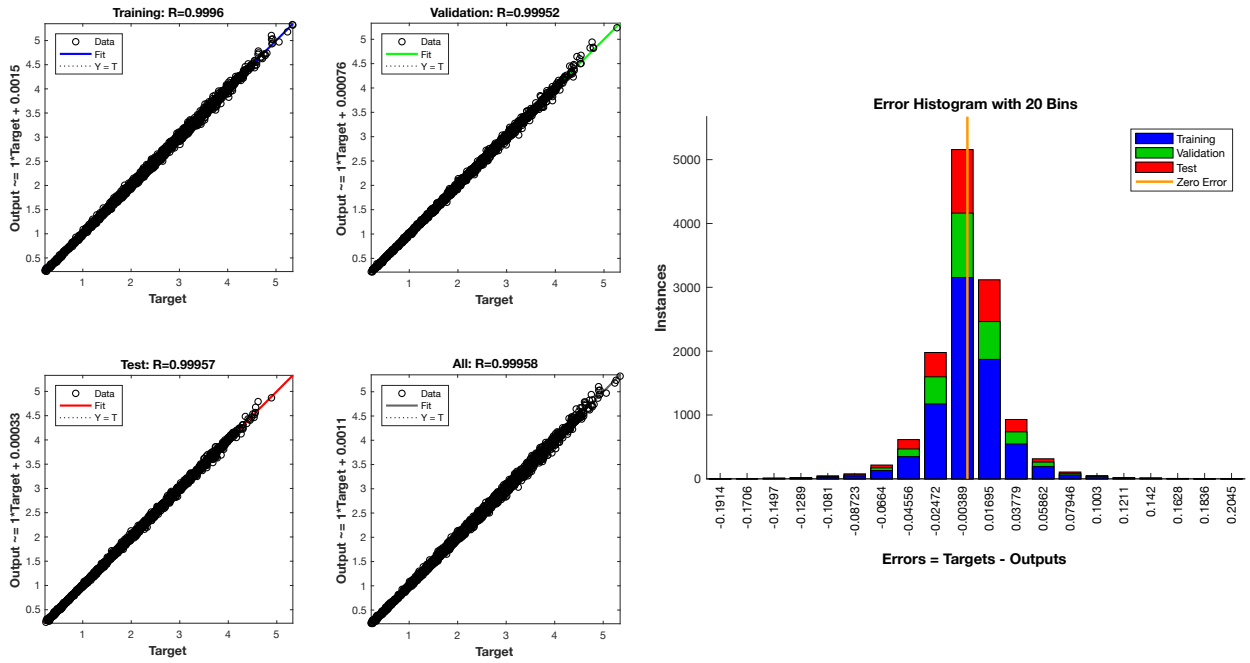


Figure 3.8: Regression plot and Error Histogram of a NN, CR=10 and HN=3

Developing the neural network: Attempt #3

Let us consider now the input matrix obtained with the features extraction compression rate set to 20. In the experiment, the time to train got unexpectedly a little higher with 44 input nodes and 5 hidden nodes.

HIDDEN NODES	TIME TO TRAIN (mm:ss)	REGRESSION VALUE
5	04:16	0.9999
3	00:06	0.99958

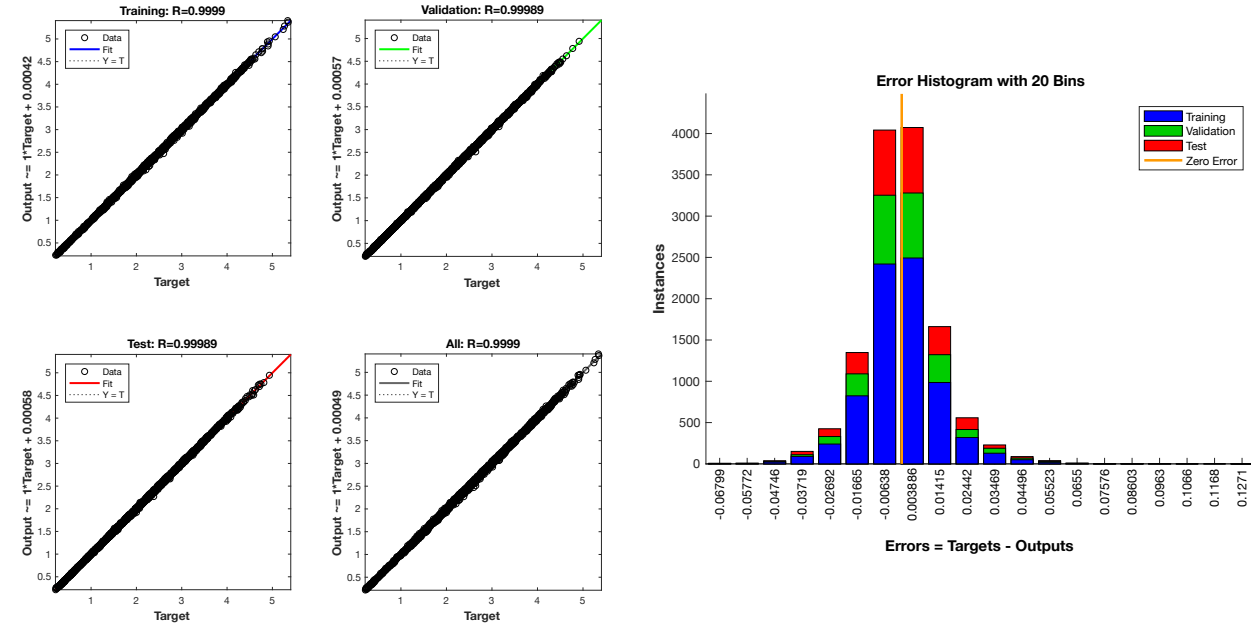


Figure 3.9: Regression plot and Error Histogram of a NN, CR=20 and HN=5

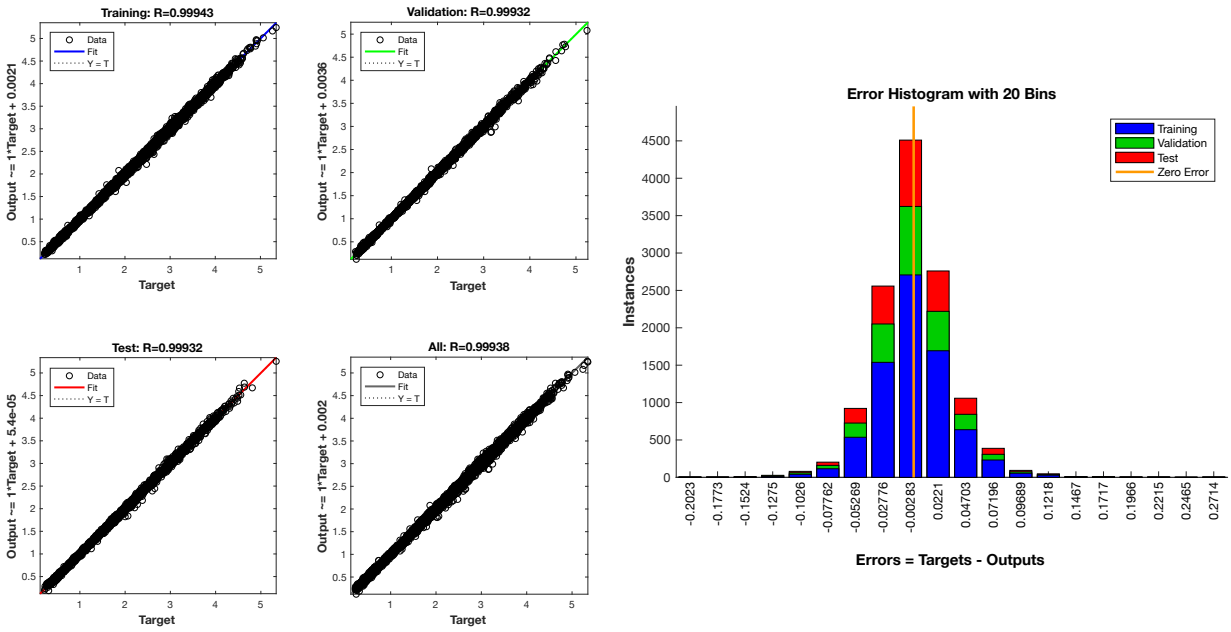


Figure 3.10: Regression plot and Error Histogram of a NN, CR=20 and HN=3

Developing the neural network: Attempt #4

Let us consider now the input matrix obtained with the features extraction compression rate set to 40.

HIDDEN NODES	TIME TO TRAIN (mm:ss)	REGRESSION VALUE
5	00:09	0.9998
3	00:02	0.998

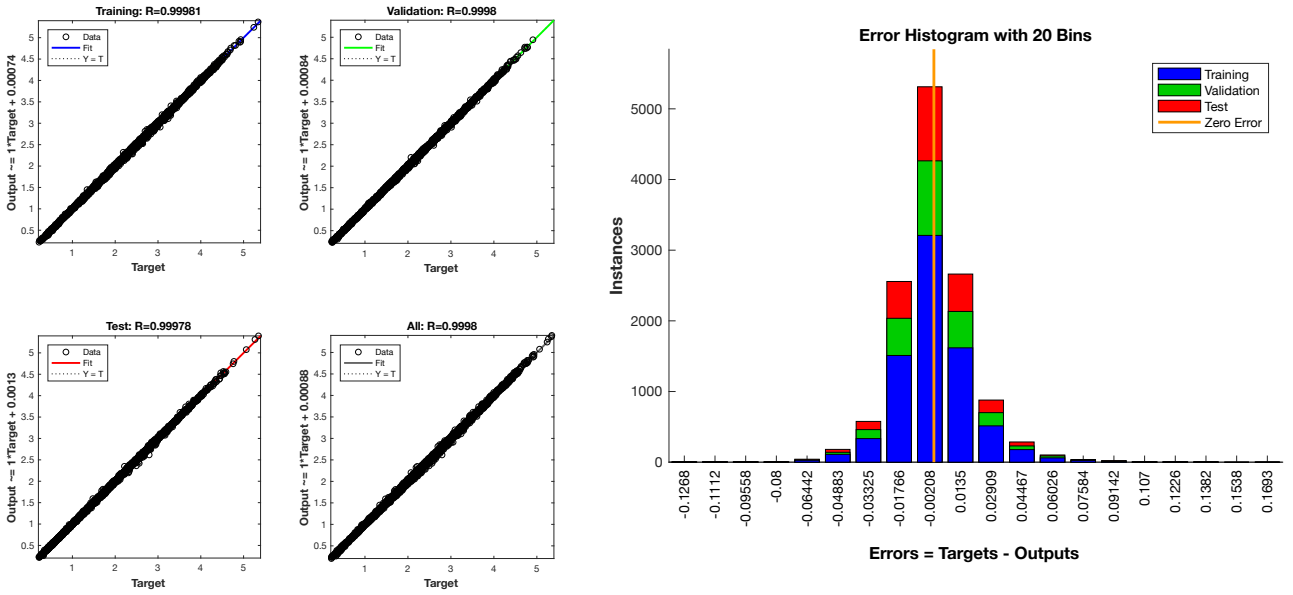


Figure 3.11: Regression plot and Error Histogram of a NN, CR=40 and HN=5

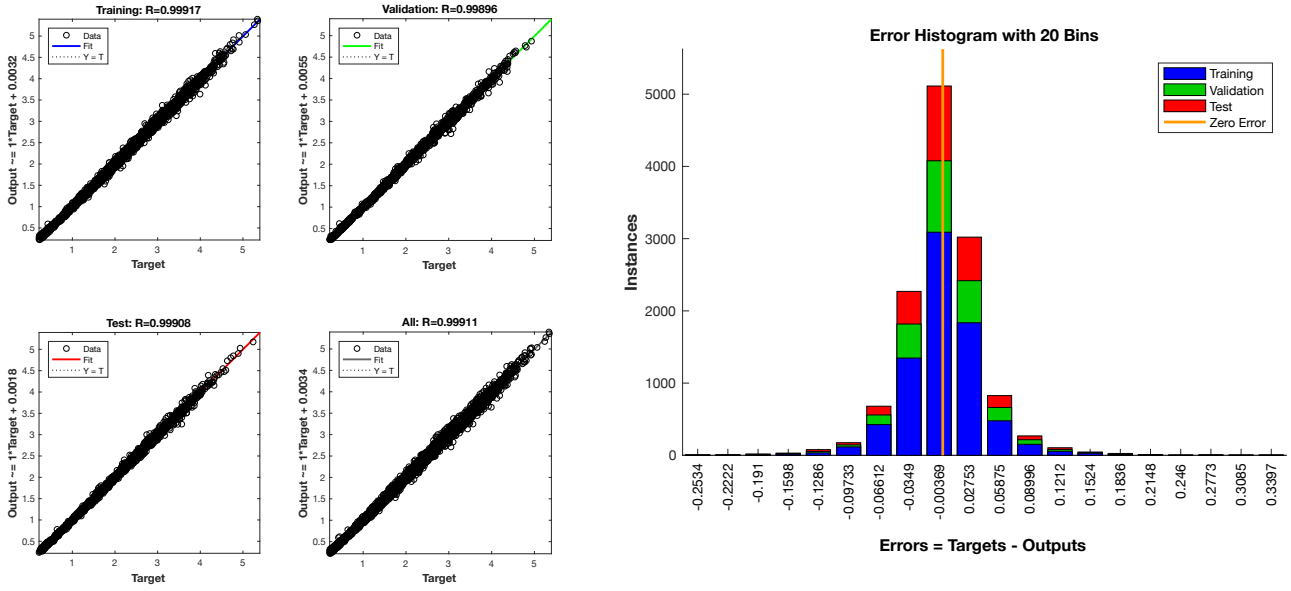


Figure 3.12: Regression plot and Error Histogram of a NN, CR=40 and HN=3

Feature selection

Even if we decided to go with 22 input nodes (which is a lot smaller than the initial 842 samples of both master and copy), it would still be too much. The features extracted previously should now be *selected* through a process of features selection.

The idea is to use the MATLAB function **`sequentialfs`**`(fun, X, y)`: selects a subset of features from the data matrix `X` that best predict the data in `y` by sequentially selecting features until there is no improvement in prediction.

```
fl = sequentialfs(@fit, input, target, 'cv', 'none', 'nfeatures', nFeatures);
features = input(:, fl);
```

- **`fit`** is the function that perform a training on the neural network;
- **`nFeatures`** is the parameter that specify how many features sequantialfs should select;
- `fl` is a vector of zeros and ones, where the ones indicates the indeces of the columns of the selected features.

Several attempts were done in order to find the best features capable to guarantee the best solution in terms of execution time and regression value; in the following table it is possible to see a recap of the experiments.

#	COMPRESSION RATE	# FEATURES	TIME TO SELECT (s)	# HIDDEN NEURONS	REGRESSION
1	20	4	373	3	0.9929
2	20	4	371	5	0.9936
3	20	6	775	3	0.99705
4	20	6	992	4	0.99747
5	20	6	738	5	0.99843
6	40	6	321	3	0.99682
7	40	6	379	4	0.99654
8	40	4	158	3	0.99298
9	40	4	189	4	0.99533
10	40	4	192	5	0.99588
11	50	4	118	3	0.99364
12	**50**	**4**	**147**	**5**	**0.99682**
13	50	6	252	3	0.99709
14	50	6	320	5	0.99831

As we can see, all the tests produced neural networks with quite good performance and regression values. As we could have expected, by changing compression rate the time required to select the features changed too; but experiments also demostrated that 9 features (*compressoin rate equals to 50*) for each signal were enough to have good results. On the other hand, increasing the numbers of hidden neurons indeed increased the accuracy of the network, but all the attempts (even with 3 hidden neurons) produced good results.

Attempt **number 12** was chosen as the best solution for the neural network in terms of performance and time to perform the features selection. The features selected are those corresponding to the column indeces **`[2,4,8,13]`** of the matrix **`[MasterExtractedFeatures CopyExtractedFeatuers]`**[4]. The *figure 3.13* shows the regression plot and error histogram for the network trained in the attempt number 12. Whereas in the *figure 3.14* it is possible to see the final scheme of the network designed to solve the first part of the project.

[4] The matrix (with 12690 rows and 18 columns) extracted in the features extraction phase with compression rate equals to 50. This also means that features 2 and 4 were extracted from the master, while features 8 and 13 were extracted from teh copy. *See chapter "Features Extraction" for more details on the input matrix.*

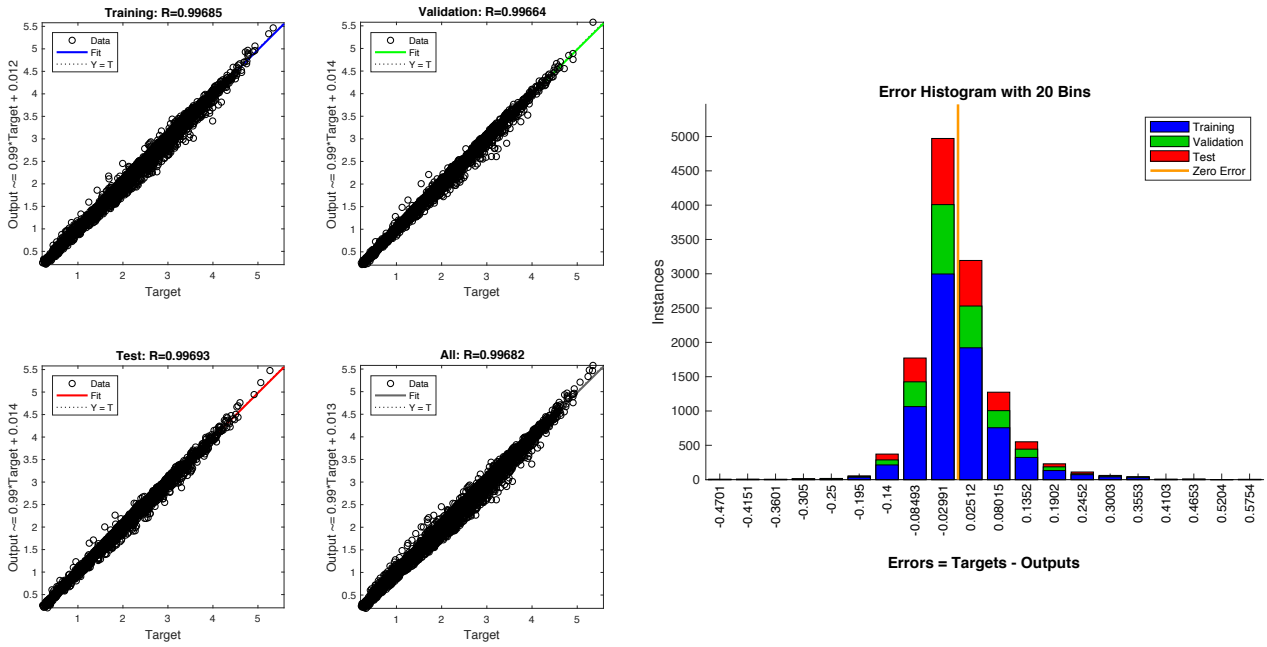


Figure 3.13: Regression plot and Error histogram for the final NN of the first part of the project

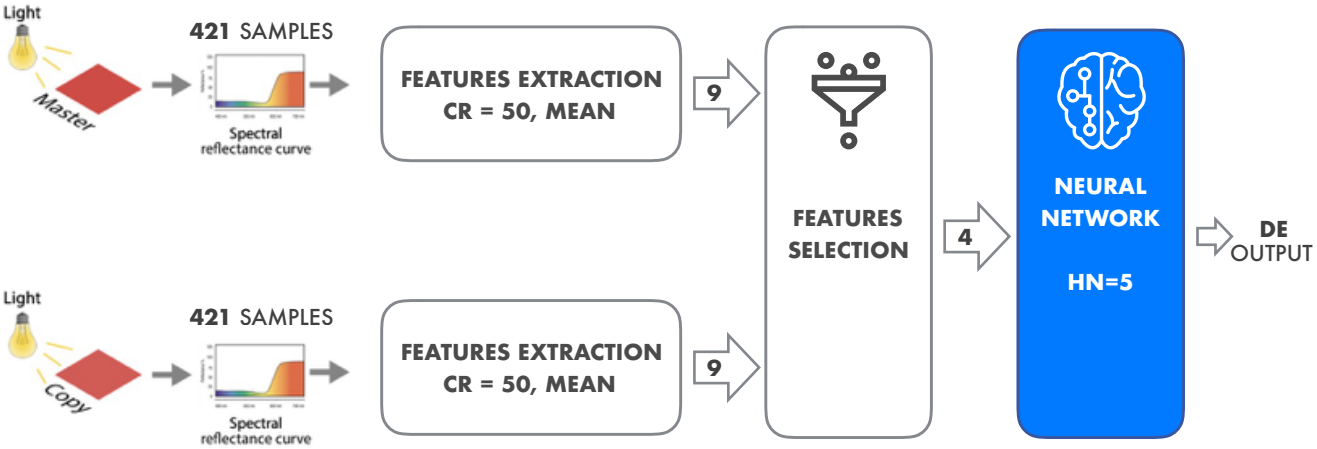


Figure 3.14: Final scheme of the neural network for the first part of the project

Part 2: Fuzzy Inference System

The CIE L*C*h* colour space and the fuzzy logic

The aim of this second part is to design and develop a fuzzy inference system to fix the deficiensies of the DE function.

The problems with the DE function

The CIE DE function defined for the L*a*b* colour space is indeed a simple equation – *the eucledian distance is nice and fast to be evaluated*. Unfortunately it is not so precise. For example there are regions where an observer perceives a difference between two colours only when they have a DE value that is much more higher than 3.5 (*according to the DE function when the difference is higher than 3.5 a clear difference in colours is noticed from both unexperienced and experienced observers*); there are also regions where an observer perceives a well recognizable differences even with small DE values[5]:

- Dark colours (hard to recognize the differences even with high values of DE);
- Blue-violet region (well recognizable differences even with small values of DE);
- Exaggerate value[6] of DE for yellowish colours;
- Exaggerate value of DE for insaturated colours;

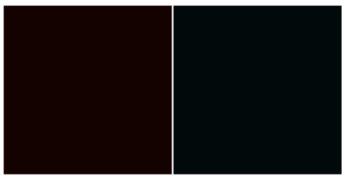


Figure 4.1: Two patches filled with dark blue and dark red colours, DE ≃ 10

Delta E provides a value indicating the overall difference between two colours. It does not provide any colour-related data such as which colour is lighter/darker, redder/greener, more blue/more yellow. To understand how the colours are different, we have to evaluate the comparative L, a, and b differences independently.

[5] W. Collins, A. Hass, K. Jeffery, A. Martin, R. Medeiros, S. Tomljanovic. Graphic Communications Open Textbook Collective (2015). *Graphic Design and Print Production Fundamentals.*

[6] With respect to the human perception of the colours.

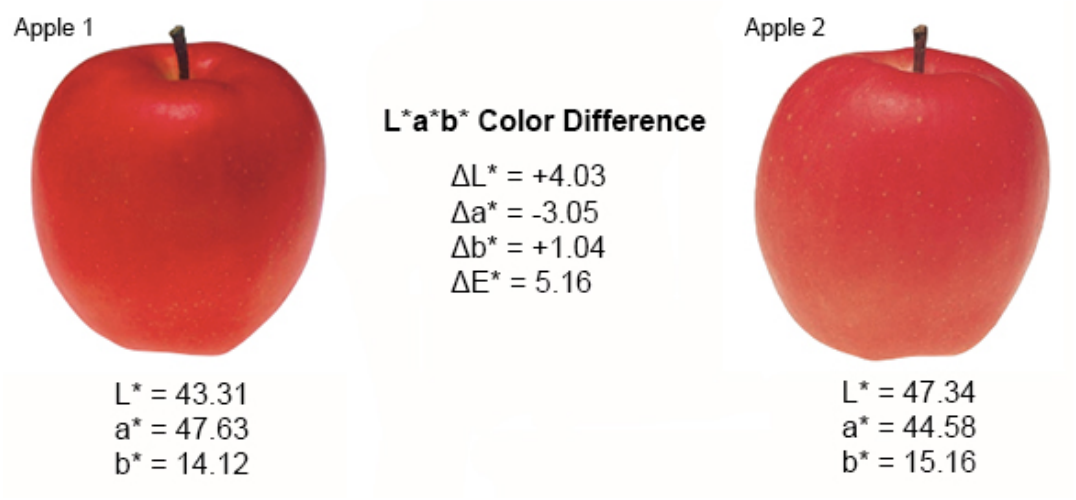


Figure 4.2: The DE function and the difference applied to just one LAB component

One of the reasons for these imprecisions is the fact that the L*a*b* colour space *is not perpetually uniformed as its creator had intended, therefore the perception of color differences changes with the location of colors within the color space and with the direction in the color space*[7]. Furthermore, the human eye is more sensitive to certain colours than others[8].

CIE L*C*h colour space

The CIE L*C*h colour space is a *perpetually uniform* colour space that represents colours by means of *lightness* (L*), *chroma* (C*) and *hue angle* (h). The CIE L*C*h colour space has the same diagram as the L*a*b* space, but uses cylindrical coordinates.

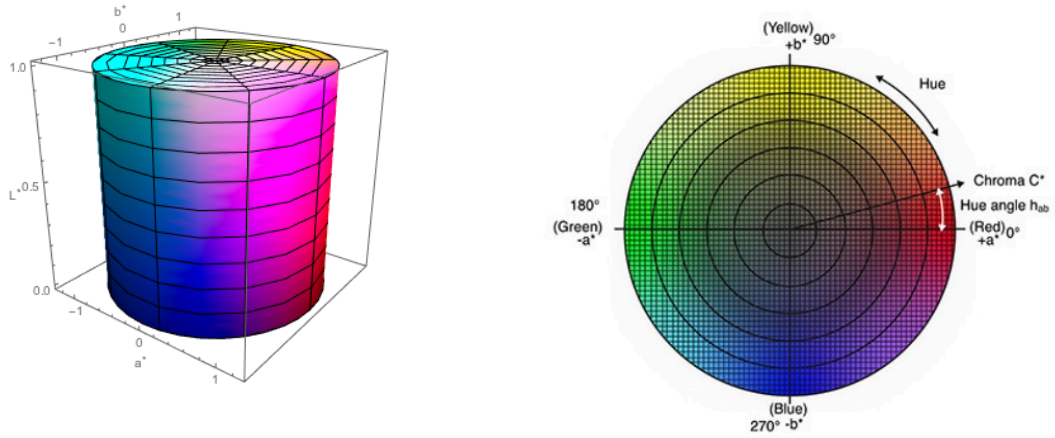


Figure 4.3: The L*C*h space (left) and a slice of the L*a*b* space at L*=50 (right)

[7] Hill, B., Roger, T., & Vorhagen, F. W. (1997). *Comparative analysis of the quantization of color spaces on the basis of the CIELAB color-difference formula.*

[8] https://en.wikipedia.org/wiki/Color_vision#Physiology_of_color_perception

The chroma is the distance from the lightness axis; the higher it is, the higher the saturation and purity.

The structure of the Fuzzy System

The neural network as it was defined in the previous chapters is not enough, we need to add something more. The idea is to use a *Fuzzy Inference System*. The fuzzy logic was introduced in the 1965 by Lofti Zadeh with the goal to provide a mechanism able to reproduce the *vagueness of the real world*. The colour comparison is a definitely a good example of vagueness, the color perception is strictly subjective and as we've already said it is sensitive to some colours rather than others. That's why is totally legit to define a fuzzy inference system to adjust the value of the difference.

The fuzzy system takes four inputs (the L*C*h coordinates and the DE evaluated from the neural network) and produces one output (the adjusted difference).

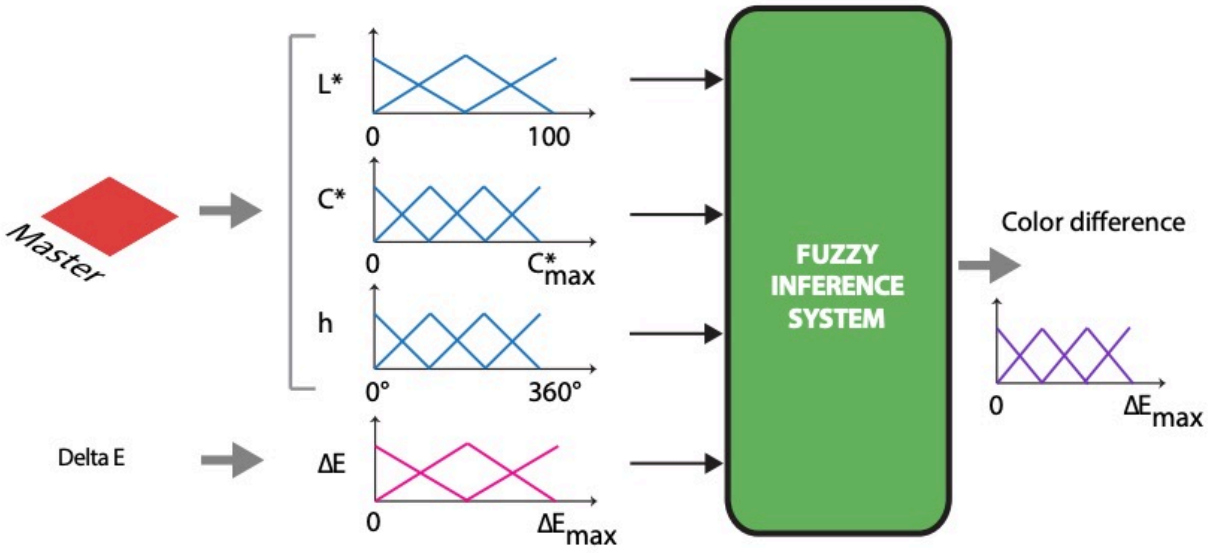


Figure 4.4: The fuzzy inference system

NOTE ON THE FUZZY INFERENCE SYSTEM
Just to be clear, it was used a *classical Mamdani Fuzzy Inference System* with the following configuration:

- **Implication**: Min function
- **Aggregation**: Max function
- **Defuzzifier**: Centroid Method

Fuzzy Sets and Membership Functions

The first thing we need to do is defining the membership functions for the input variables: the *HUE angle*, *lightness value*, *chroma value and DE*.

HUE

The HUE value is *the degree to which a stimulus can be described as similar to or different from stimuli that are described as red, green, blue, and yellow*[9].

The HUE variation can be translated into a fuzzy set in range **[0, 360]** with the following membership functions:

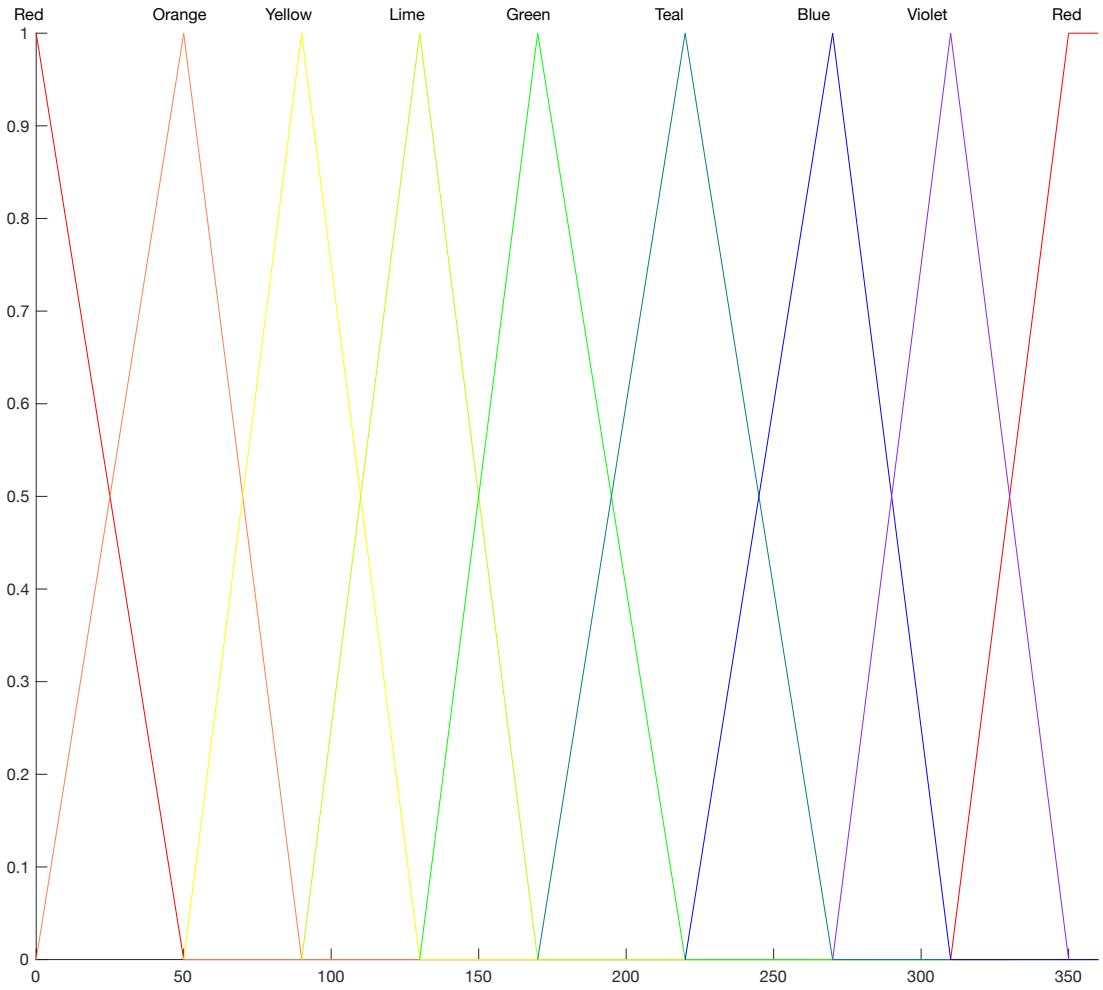


Figure 4.5: Fuzzy Set for HUE

- **Red:** [50 310 350 360];
- **Orange:** [0 50 90];
- **Yellow:** [50 90 130];
- **Lime:** [90 130 170];
- **Green:** [130 170 220];
- **Teal:** [170 220 270];
- **Blue:** [220 270 310];
- **Violet:** [270 310 350];

[9] https://en.wikipedia.org/wiki/Hue

Since there is no standard membership function to describe the colour red it was defined a custom function:

```
function out = customFuzzyRed(x, params)
      for i = 1:length(x)
          if x(i) < params(1)
              y(i) = -x(i)/params(1) + 1;
          elseif x(i) < params(2)
              y(i) = 0;
          elseif x(i) < params(3)
              y(i) = (x(i)-params(2))/(params(3)-params(2));
          elseif x(i) <= params(4)
              y(i) = 1;
          end
      end
      out = y';
end
```

Lightness

According to CIE standards, lightness is a percentage value ranging from 0 to 100. The membership functions for luminosity are:

- **Dark:** [0 0 10 20];
- **Medium:** [10 20 80 90];
- **Bright:** [80 90 100 100];

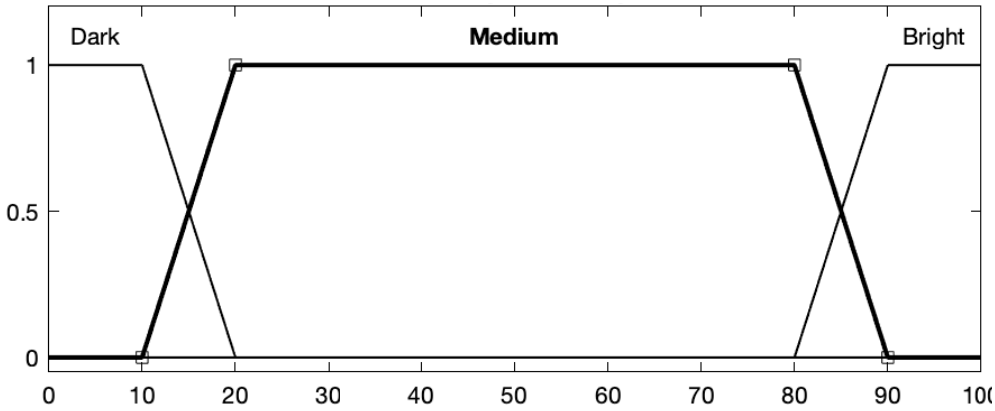


Figure 4.6: Fuzzy Set for Lightness

Chroma

Chroma should be a percentage value (**[0 100]**), things are not as easy as the previous fuzzy sets though. The maximum values for chromacy at each slice (*the chromacy values are obtained through a coordinates transformation from a spherical colour space to a cylindrical colour space*), depend on the value of the lightness. Since we are going to use the same system for any *slice* (*we want a uniform percentage*) of the space it is necessary to *evaluate* the maximum values of chroma at each slice. In order to make it

easier we can just assume the maximum value for C* (*127, at L*=50*[10]) and then, by applying simple geometric formulas we obtain that the relation between chroma and lightness can be approximated using the equation of an ellipse. So, starting from a given L*C*h colour coordinate we have that the maximum value of C for that specific slice is:

$$C_{MAX} = 127 * \sqrt{1 - (\frac{L - 50}{50})^2}$$

and then we can simply obtain the percentage of chroma with respect to the C_{MAX}as:

$$C_{\%} = 100 * C / C_{MAX}.$$

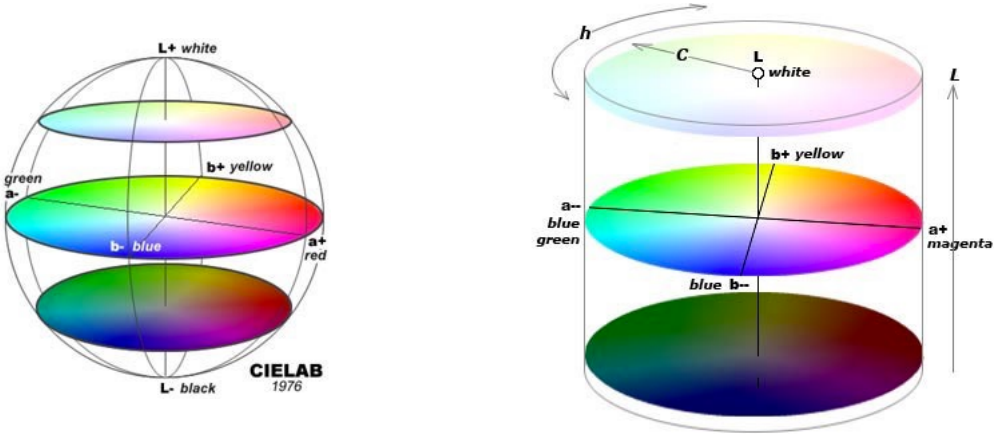


Figure 4.7: CIE L*a*b* sphere and CIE L*c*h cylinder comparison

After that it is possible to obtain the following membership functions:

- **LowSaturation:** [-0.35 50];
- **HighSaturation:** [0.35 50];

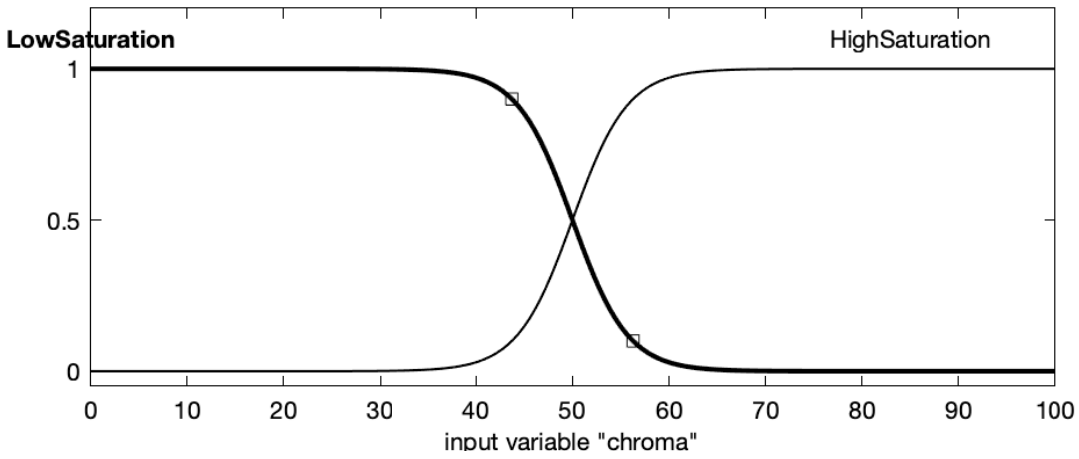


Figure 4.8: Fuzzy Set for Chroma

[10] Actually, this is an approximation: C^*_{MAX} is a number between 127 and 128.

DE and Adjusted DE

The last fuzzy set is for the values of DE (*evaluation of the CIE DE 76 colour difference function for all the master-copy pairs*). The same membership functions were used for both input DE and Adjusted output. DE values vary in **[0 7]**.

- **Same:** [0 0 1.25];
- **Expert:** [0.5 1.25 2];
- **NonExpert:** [1.25 2.5 3.75];
- **Visible:** [3 4 5];
- **Different:** [6.84 4.54285714285714];

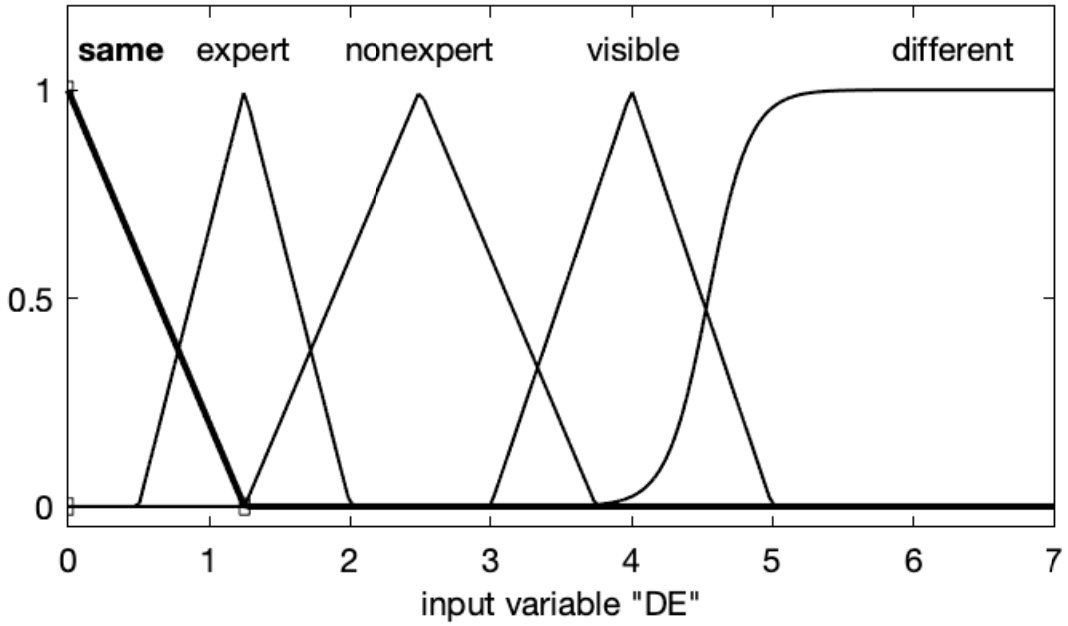


Figure 4.9: Fuzzy Set for DE and $DE_{Adjusted}$

Fuzzy Rules

To complete the Fuzzy Inference System it is necessary to define some rules for the regions we want to adjust: the desaturated colours, the dark colours, the blue-violet area and the yellowish colours.

Here are reported the most significant rules for the developed Fuzzy Inference System:

The following rules were generated based on prior knowledge about the problem and colour theory.

```
if (L is Dark)
then (DEadj is same) (1)

if (L is Medium) and (C is LowSaturation) and (DE is expert)
then (DEadj is nonexpert) (1)

if (L is Medium) and (C is LowSaturation) and (DE is nonexpert)
then (AdjustedDE is visible) (1)

if (L is Medium) and (C is HighSaturation) and (H is Yellow) and (DE is
nonexpert)
then (DEadj is expert) (1)

if (L is Medium) and (C is HighSaturation) and (H is Yellow) and (DE is
visible)
then (DEadj is nonexpert) (1)

if (L is Medium) and (C is HighSaturation) and (H is Yellow) and (DE is
different)
then (DEadj is nonexpert) (1)

if (L is Medium) and (C is HighSaturation) and (H is Blue) and (DE is expert)
then (DEadj is nonexpert) (1)

if (L is Medium) and (C is HighSaturation) and (H is Blue) and (DE is
nonexpert)
then (DEadj is visible) (1)

if (L is Medium) and (C is HighSaturation) and (H is Violet) and (DE is
expert)
then (DEadj is nonexpert) (1)

if (L is Medium) and (C is HighSaturation) and (H is Violet) and (DE is
nonexpert)
then (DEadj is visible) (1)
```

Training the Fuzzy Inference System

For each master-copy pair there has been an input for the fuzzy system composed by the LCH coordinates of the master patch and the value of DE (*for the i-th master-copy pair*) evaluated using the standard CIEDE76 function.

Figure 4.9: Yellow area fuzzy correction – DE = 4.056, DE_{Adj} = 3.251

Figure 4.10: Blue-Violet area fuzzy correction – DE = 2.452, DE_{Adj} = 3.155

Figure 4.11: Low saturation fuzzy correction – DE = 1.999, DE_{Adj} = 3.152

Figure 4.12: Low saturation fuzzy correction – DE = 2.001, DE_{Adj} = 3.153

Putting all together

The final step of the second part of the project is trying to exploit the Fuzzy Inference Systems's results in order to correct the neural network's behaviour.

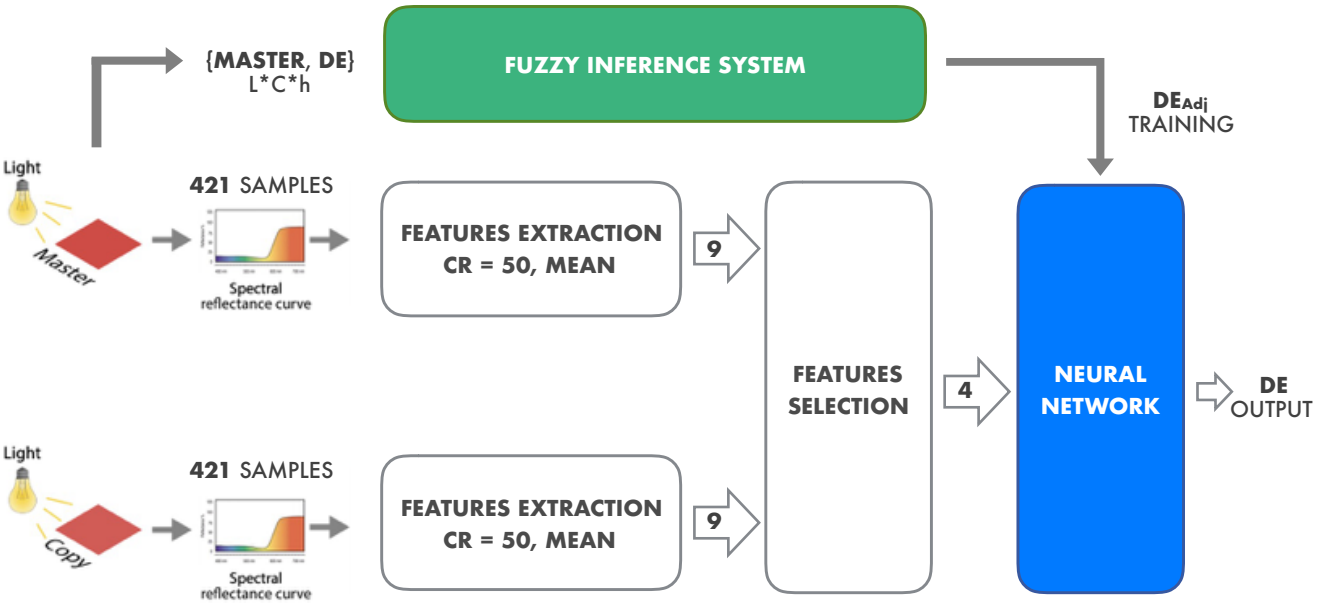


Figure 4.13: Final schema of the Neural Colour Comparison System

The configuration for the training and feature selection is the same as the one used in the first part:

- **Data division**: Random, with *60% training, 20% testing, 20% validation*;
- **Training**: Levenberg-Marquardt;
- **Performance**: Mean Squared Error;
- **Features Extraction Function**: Statistical Mean (*one every **50** samples)*;
- **Number of selected features**: 4;

#	# HIDDEN NEURONS	TIME TO SELECT (s)	REGRESSION	TIME TO TRAIN (s)
1	2	112	0.95768	6
2	3	160	0.97156	1
3	4	130	0.97761	2
4	5	144	0.96884	3
5	**6**	**170**	**0.98118**	**1**
6	7	186	0.97535	2
7	10	246	0.98086	5
8	25	697	0.99159	21

As it is shown in the table, there has been a decline in the performance. At least 25 hidden neurons are needed in order to provide the same regression value as the one obtained in the first part of the project. However, 25 neurons are too much and they require a lot of time to perform features selection and training. That's why attempt number five was chosen as the best result. In figure 4.14 there are shown the regression plot and error histogram.

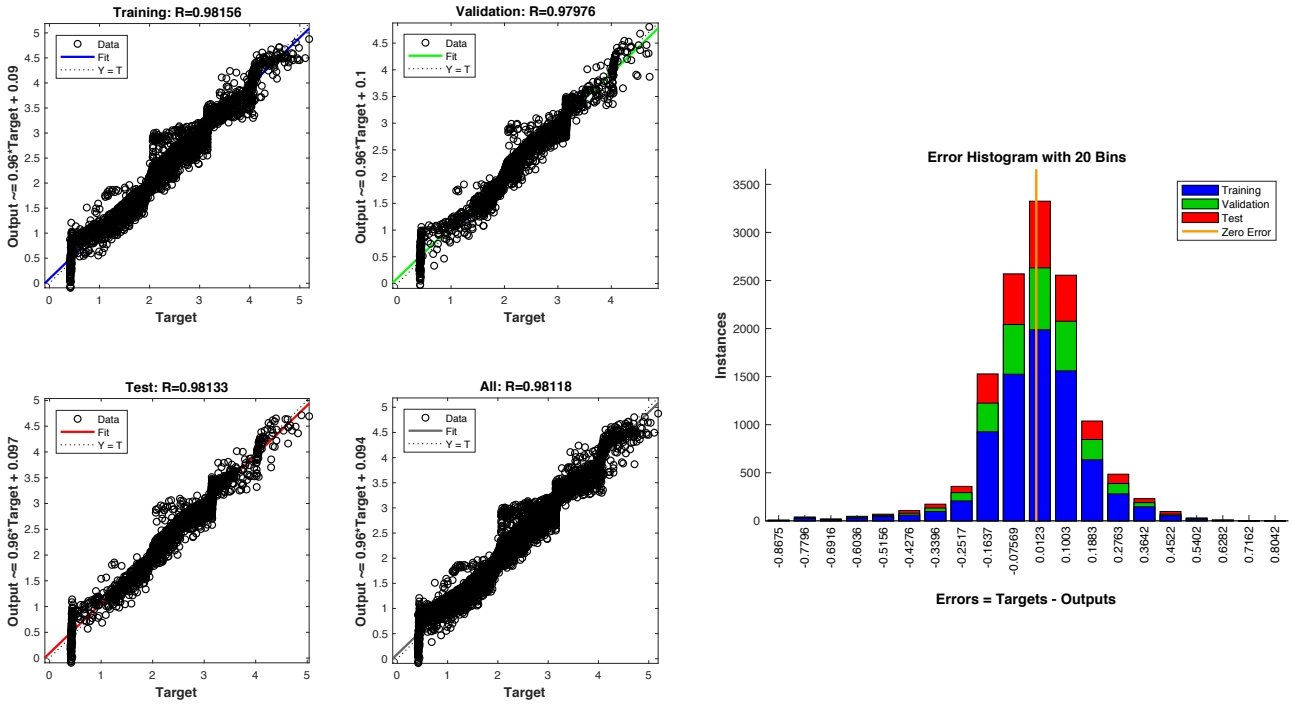


Figure 4.14: Regression plot and Error histogram for the final NN of the project

Conclusion

So, what now?

In the first part of the project a Function Fitting Neural Network was designed and implemented in matlab; the network is able to reproduce the behaviour of the CIE-DE-76 function by evaluating a selected set of features (statistical mean) of two colours spectra (one for the master patch and one for its copy patch). There has been satisfying results in both efficiency and accuracy, however the DE function used to develop the Neural Network has got some problems with specific colours. To get over these limits, in the second part a Mamdani Fuzzy Inference System was added to the Neural Colour Comparison System. The DE values were adjusted using fuzzy logic and then were used to train the network from the first part of the project. Results were not as good as the ones achieved in the first part, but still satisfying for our aims.

Appendix

Source Code

You can find the full source code for the project at http://tinyurl.com/ISScotto or you can just scan the QR Code.

List of figures