Computational Intelligence: Report

Ciardi Roberto, Degiovanni Alessandro, Falzone Giovanni

Embedded Computing Systems, Sant'Anna School of Advanced Studies. Pisa, Italy

January 11, 2018

Contents

1 Introduction

This report explains the work made to build a neural network, to recognise colours' differences. The main purpose of the project is firstly described, followed by few words about colours and perception, then, in the first part of the document, we talk about the work made on dataset, to have a set of colours suitable for our work. After dataset building, we introduce a study of **L*a*b*** space and the way in which we derive an enhancement of ΔE, the measurement of distance between **L*a*b*** colours. This part was important because it allowed us to have an improved measurement of colour differences, based on human perception. In the second part of the document we describe the MATLAB tool that we created, in order to work in an easier way on colours. Finally the process of building, training and testing of our Neural Networks is explained, with particular attention to the building of training set and to the implementation choices that took to our final results.

1.1 Purpose of the project

The aim of the project is to design and develop a neural network-based system to compare two colour patches and measure their similarity. Given a colour patch called *master*, which must be faithfully reproduced by an industrial printing process, the quality requirement of this process is that every reproduction of the master - called *copy* - is chromatically equal to the master, when the patches are observed under the same light source. So our purpose is to give a precise measurement of the differences between a master patch and its copies, by means of a network that must distinguish the colours and understand how humans perceive them.

1.2 Colours and Perception

Colour is a physiological sensation that our brain uses to recognize the objects around us. The colour sensation occurs according to four steps: a light is emitted by a source, the light is reflected by an object surface, the human eye detects the reflected light that becomes a stimulus for the brain.

So the light has a leading role in colour recognising process made by human eye: the light reflected by an object has a *Spectral Power Distribution* (**SPD**), referred as *reflected SPD*, that is obtained multiplying the energy of the light source by the reflectance of the object.

A colour is almost always a blend of different wavelengths of lights: given the **SPD** of a light that reaches eye's retina, three types of cones generate a tuple of three numbers, called *tristimulus response*. Given S(λ) as the **rSPD**, with λ wavelength, and $\bar{\rho}(\lambda)$, $\bar{\gamma}(\lambda)$ and $\bar{\beta}(\lambda)$ as the sensitivity functions of the three types of cones, the tristimulus response can be computed by integrating the products of wavelength intensity and cone sensitivity over the visible spectrum:

$$\rho = \int_{\lambda_l}^{\lambda_u} \mathrm{S}(\lambda)\bar{\rho}(\lambda)\,\mathrm{d}\lambda$$

$$\gamma = \int_{\lambda_l}^{\lambda_u} \mathrm{S}(\lambda)\bar{\gamma}(\lambda)\,\mathrm{d}\lambda$$

$$\beta = \int_{\lambda_l}^{\lambda_u} \mathrm{S}(\lambda)\bar{\beta}(\lambda)\,\mathrm{d}\lambda$$

Our colour perception is determined by this 3-tuple response, with $\bar{\rho}(\lambda)$, $\bar{\gamma}(\lambda)$ and $\bar{\beta}(\lambda)$ that are called *colour matching functions* and that can be used to represent any visible colour.

1.3 CIE RGB and CIE L*a*b* colour spaces

To represent colours in computer graphics, some colour models are given as **RGB** and **L*a*b***, that have been standardized by **CIE** (*Commission Internationale de l'Eclairage*) and that are the ones used in our work.

CIE RGB(*Red Green Blue*) is a convenient model as the human visual system works in a similar way to an **RGB** colour space. The 3-tuple (R,G,B) that describes a colour is obtained as

$$(R,G,B) = \left(\int_{\lambda_l}^{\lambda_u} \mathrm{S}(\lambda)\bar{\mathrm{r}}(\lambda)\,\mathrm{d}\lambda), \int_{\lambda_l}^{\lambda_u} \mathrm{S}(\lambda)\bar{\mathrm{g}}(\lambda)\,\mathrm{d}\lambda), \int_{\lambda_l}^{\lambda_u} \mathrm{S}(\lambda)\bar{\mathrm{b}}(\lambda)\,\mathrm{d}\lambda) \right)$$

where $\bar{r}(\lambda)$, $\bar{g}(\lambda)$ and $\bar{b}(\lambda)$ are the colour matching functions introduced to represent a wider set of colours, each in terms of an amount of red, green and blue light. Given a light source, the 3-tuple (R,G,B) uniquely determines a visible colour.

The **CIE L*a*b*** is a different colour space introduced as a perceptually uniform space, where the difference of two colours, sampled with a given distance in the space, is perceived the same way, no matter the regions they come from. This space is used when graphics to print are converted from **RGB** to **CMYK** - *Cyan Magenta Yellow Key-black* used by printers -, as the **L*a*b*** gamut includes both the **RGB** and **CMYK** gamuts.

The **L*a*b*** space describes all the perceivable colours in three dimensions: **L*** for lightness, **a*** and **b*** for the colour opponents green-red/magenta and blue-yellow, respectively. In particular, **L*** value is included in [0, 100], with the largest values that are the clearest; **a*** is included in [-127, 127], with negative values that indicate green colours, while positive indicate magenta; **b*** is included in [-127, 127], with negative values that indicate blue and positive values that indicate yellow.

2 Dataset

The given dataset is made up of the reflectance spectra of 1269 matt colour patches, measured with **1nm** step, ranging from **380nm** to **800nm**. This results into 421 samples for each reflectance curve. Two matrices are given, named `spectra` and `coordinates`:

- `spectra` is a 421×1269 matrix where each column corresponds to a colour patch and each row to one of the 421 samples of the reflectance curve of that colour patch;
- `coordinates` is a 6×1269 matrix where each column corresponds to a colour patch, rows from 1 to 3 contain the **RGB** coordinates in the order, and rows from 4 to 6 contain the **L*a*b*** coordinates in the order. The colour coordinates are calculated using **D65** light source - the daylight, **6500°K** -.

The given data were not suitable for our work, for example we had **RGB** values not included in [0,1] interval, so we reviewed them to obtain a most suitable dataset.

2.1 Function computecolours

The function `computecolours`, contained in *computecolours.m* file, computes the new dataset, returning three matrices: rgb, lab and new_spectra. We used some of the functions provided by the *Image Processing Toolbox* by MATLAB. We calculated the **XYZ** coordinates - used to represent colours in one of the first colour space, mathematically defined by **CIE** - of the spectra given in our dataset. We know that we can find the values for a colour with an **SPD** defined as I(λ) by these integrals:

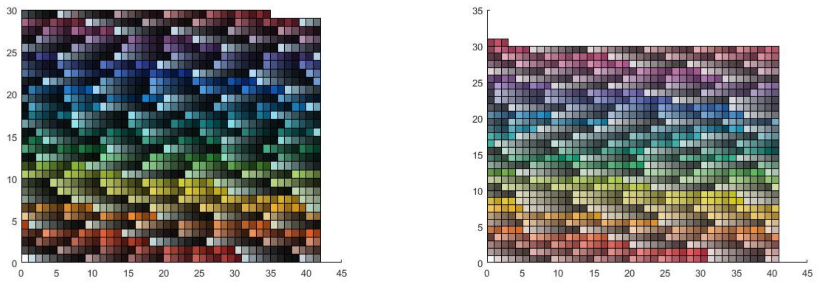
$$X = \int_0^\infty \mathrm{I}(\lambda)\bar{\mathrm{x}}(\lambda)\,\mathrm{d}\lambda$$

$$Y = \int_0^\infty \mathrm{I}(\lambda)\bar{\mathrm{y}}(\lambda)\,\mathrm{d}\lambda$$

$$Z = \int_0^\infty \mathrm{I}(\lambda)\bar{\mathrm{z}}(\lambda)\,\mathrm{d}\lambda$$

So we calculated the values of **XYZ** colour matching functions ($\bar{x}$, $\bar{y}$, $\bar{z}$) with function `colourMatchFun(1931_FULL)`. Then we used `illuminant('d65')` function to have the spectral power density at each wavelength, for the illuminant D65, defined as the daylight. So we had the matrices `xbar`, `ybar`, `zbar` and `energy`, and we excluded the values outside visible spectrum, so we took only value with λ in the interval [380,800].

Starting from the given spectra we calculated the reflected spectral power distribution by multiplying `spectra` matrix with the energy returned by `illuminant` function. Then we approximated the integral as a multiplication, and we calculated X,Y and Z coordinates of our **rSPD** by multiplying this, by `xbar`, `ybar`


(a) Original Dataset

(b) Generated Dataset

Figure 1: Colours of original and generated datasets

and `zbar`. Finally we normalized the **XYZ** coordinates, dividing by a normalization factor `k` calculated as `ybar` multiplied by `energy`.

This work allowed us to have the **XYZ** coordinates of our space, then, by using `xyz2lab` and `xyz2rgb` functions, we obtained the new **L*a*b*** and **RGB** coordinates of our colours. We excluded the colours that gave negative values for RGB coordinates - 37 of them - so we finally had 1232 colours. These colours are resumed in three matrices:

- `rgb` is a 3×1232 matrix, containing the **RGB** coordinates of our colours.
- `lab` is a 3×1232 matrix, containing the **L*a*b*** coordinates of our colours.
- `new_spectra` is a 421×1232 matrix, containint the spectral power distributions of our colours.

The new dataset, used for our work, is contained in `NEW_dataset.mat` workspace.

2.2 Differences

As said before, this work was useful to discard non-suitable values of the original dataset and to have **RGB** coordinates included in [0,1] interval. We can see the difference between the colours of the two datasets, drawing the patches relative to each colour. We used the function `drawPatches()`, contained in fil `drawPatches.m` to have two figures, representing the datasets. This function draws some squared patches side by side, coloured with the **RGB** colours passed as argument: for each **RGB** thriple it controls if it is valid - between 0 and 1 - and then it draws a squared patch with the function `patch()`, given the x and y coordinates of the vertex and the colour. The result is a set of patches side by side, aligned by the optional `align` argument, that can be given to the function, not valid patches are excluded by the drawing. So we can see the differences between the two datasets in the Figure 1: we can see that the generated dataset has the same colours of the original one but clearer than the previous, this underlines that the new data are not far from the original one, but are more useful for our works.

3 The L*a*b* Space

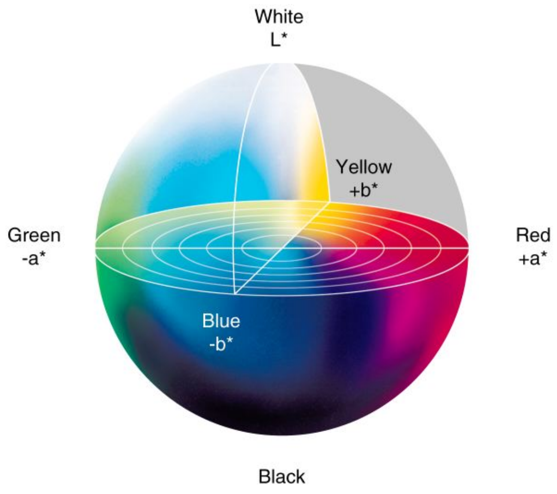


Figure 2: **L*a*b*** Space

As said before, the **L*a*b*** colour space describes mathematically all perceivable colours in the three dimensions **L** for lightness and **a** and **b** for the colour opponents green-red/magenta and blue-yellow. This space is representable as the sphere seen in figure 2. The vertical axes of the sphere represents the lightness of the colour from a minimal value of 0 (the darkest), to a maximal value of 100 (the clearest). The spherical shape of this space allows to calculate colours' differences with a precision that is near to human perception, with the nonlinear relations for L*, a*, and b* that are intended to mimic the nonlinear response of the eye. Uniform changes of components in the **L*a*b*** colour space aim to correspond to uniform changes in perceived colour. So the perceptual differences between any two colours can be approximated by treating each colour as a point in a three-dimensional space (with three components: L*, a*, b*) and taking the Euclidean distance between them. This allow us to calculate differences between two colours 1 and 2, with coordinates $(L^*{}_1, a^*{}_1, b^*{}_1)$ and $(L^*{}_2, a^*{}_2, b^*{}_2)$ respectively, with the formula:

$$\Delta E_{12} = \sqrt[2]{(L^*{}_2 - L^*{}_1)^2 + (a^*{}_2 - a^*{}_1)^2 + (b^*{}_2 - b^*{}_1)^2}$$

3.1 Different Perceptions

The distance calculated before - Lab Distance or ΔE - gives the differences between two colours as it is perceived by the human eye: two colours with a ΔE value less than 2 can be considered equals, whereas two colours with a major ΔE will be increasingly different. Unfortunately this paradigma is not valid for each colours' couple: in fact, in some zones of the sphere, colours differences are less perceivable than in others. If we consider the **L*a*b*** sphere as a sovrapposition of circles, as in figure 3, we can understand this problem, because we can see that, changing L* value, colour differences change. For example, with an L value

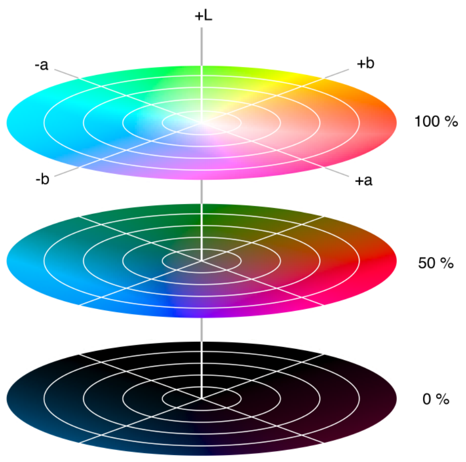


Figure 3: L*a*b* Space as Circles sovrapposition

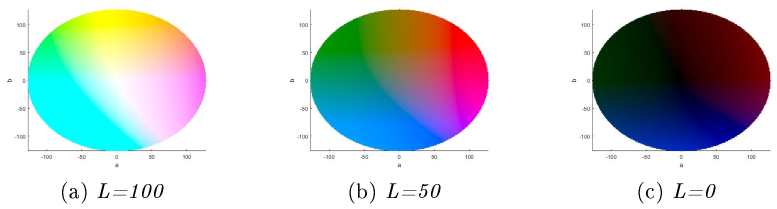


Figure 4: Circles generated with MATLAB

of 0 - dark colours - we can see that the differences are less perceivable, so, two colours with a Lab distance larger than 2, could however be considered equal. We built a MATLAB function, `generateCircle(L)`, useful to draw different sections of the **L*a*b*** sphere, in order to underline these differences. This function draws a 2D circle for the specified L coordinate value, as we can see in figure 4, where the three circles with the greatest, smallest and median L are shown. Also in this case we can see, for example, that colours in circle with low Luminosity, are very dark and so can be perceived as equal even if they have an Euclidean distance larger than 2.

So, in order to compare two colours we can use the **L*a*b*** distance, but with some differences with respect to only computing the Euclidean distances.

3.2 Modifications

We decided to calculate **L*a*b*** distance in a different way, according to the **CIE** one, but with some modification, that could improve ΔE precision. We built the function `labDistancemod(colour1,colour2)` that, given a couple of colours, returns their Lab distance with the explained modifications, according to human perception. Given the two colours, we calculate the distance on L* axis with an integral:

$$dist_L = \int_{L^*_{colour1}}^{L^*_{colour2}} \text{distance\_L(t)d}t$$

with the function `distance_L()` that is a piecewise linear function, as we can see from figure 5.

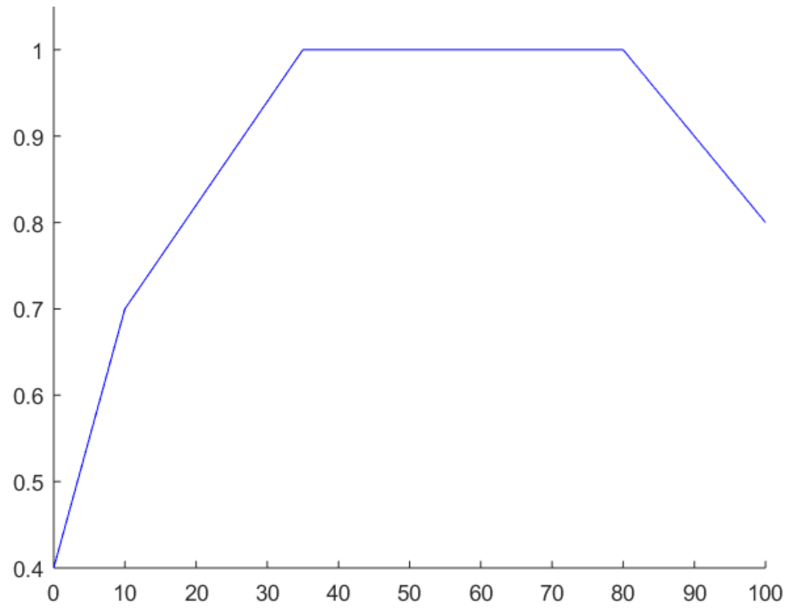


Figure 5: distance_L function raffiguration

As we can see, the values are always less or equal than 1: a greater value represents a better sensitivity - differences are more perceivable -, whereas a smaller value represents a worse sensitivity. These values are used to compute the integral shown above. The value obtained by the integral is then used to calculate the labDistance, with the known formula, using $dist_L$ as the difference between $L^*_{colour1}$ and $L^*_{colour2}$, whereas differences between the other coordinates are calculated in the same way as before. We can notice that, if all the values integrated are equal to 1, nothing changes with respect to the original Lab distance, as we expect for colours differences that are already calculated in a precise way with ΔE.

To add more precision to the calculation, the result is then multiplied by a value that represents the human luminosity perception. This value is calculated by function `luminosity_perception()`, contained in file *luminosity_perception.m*, that, given an **L*** value - in this case the average between the L* coordinates of the two colours - returns a factor that resumes the luminosity perception of that **L*** value. Also the function `luminosity_perception()` is a piecewise linear function, as we can see from the figure 6.

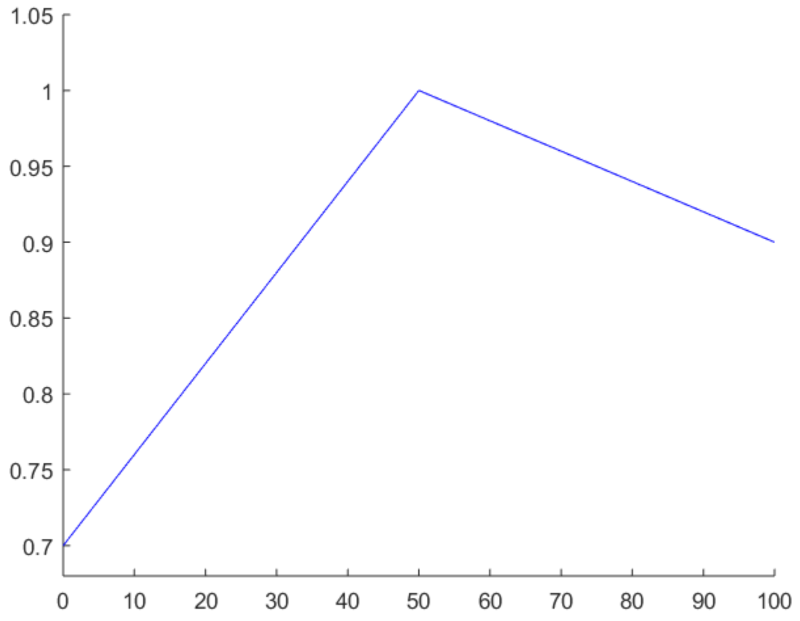


Figure 6: luminosity_perception function raffiguration

Thus, with these modifications, we have a **L*a*b*** distance that is closer to human perception than the original. We are going to see that we decided to introduce an *equality rate*, for a certain couple, that can be decided by an user with a vote, to add a more precise perception factor to couple of similar colours.

4 MATLAB Tool

In order to work with the given colours in an efficient way, we developed a MATLAB tool, with a simple graphical interface. It allows user to have a perspective of the given colours and choose some of them, to make perturbations and obtain copies. The main purpose of the tool is to allow user to generate set of couples - master, copy - to be used to train and test neural networks. The tool allows also to make comparision between colours, in order to vote them, to introduce human perception in our data.

Each **GUI** - Graphical User Interface - was created on MATLAB and is available in *GUIs* folder. These GUIs allow user to interact with our systems, by clicking buttons, that execute callback functions we developed, and that are available in *GUIs* and *GUI_Functions* folder. The data created are all saved in the current workspace or can be stored in *Variables* folder.

4.1 Main Menu

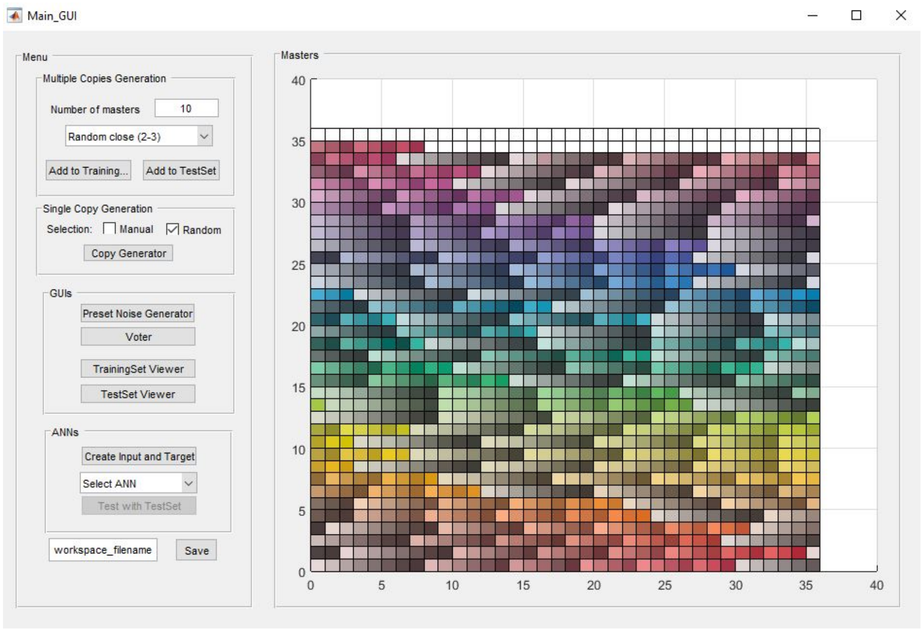

Number of masters 10
Random close (2-3)
Selection: Manual [✓ Random]
workspace_filename

Figure 7: Main Menu screenshot

In the main menu(figure 7), we have, on the right, a visualization of all the given masters, as we saw in figure 1. On the left side we have some buttons, divided in four parts, to give commands to the tool:

- The first panel - **Multiple copies generation** - is useful to generate, simultaneously, multiple couples master-copy, with a predefined criterion;
- The second panel - **Single copy generation** - is useful to generate the copy of a certain master with a certain white gaussian noise;
- The third panel - **GUIs** - gives access to some other useful interfaces;

- The fourth panel - **ANNs** - is useful to test the Neural Networks on a generated TestSet.

4.2 Multiple copies generation

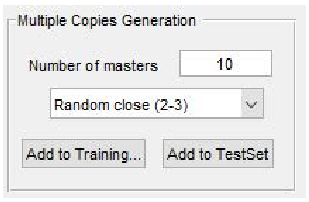


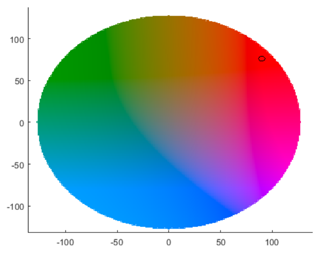
Figure 8: Multiple copies generation Menu

Here you can decide, by inserting a number in the space, how many masters you want to generate copy of. The drop-down menu allows to decide how to generate the copy for each single master, and you can decide to generate copy that are, in the **L*a*b*** colour space:

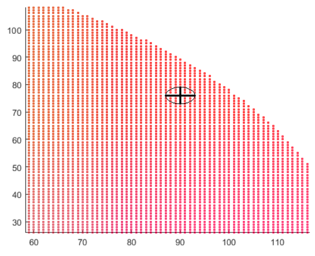
- very close to the master with a ΔE between 0.5 and 2, so a copy almost equal to the master;
- close to the master, with a ΔE between 2 and 3, these are copies that can be equal or very similar to the master;
- a little far from the master, with a ΔE between 3 and 5, these copies are usually different from the master, but equals in specific cases;
- or away from the master, with a ΔE larger than 5, that usually represent copies very different from the masters.

The algorithm chooses a number of masters equal to the inserted number, from the masters of our dataset - showed on the right of the interface - and generates four copies for each master, with every copy that is distant from the master as specified by the decision made on the drop-down menu. In particular if the whole dataset is selected (1232 masters), the algorithm generates four copies for each master, whereas, if a number less than 1232 is chosen, the masters are randomly selected: for each random masters the algorithm controls if it has been already chosen, in this case it choose another master. These control is made at most 100 times, in order to have all different randomly chosen masters.

For each master, four copies are generated, generating a random number between -1 and 1, multiplying it by the maximum value of ΔE chosen. In this generation we consider a sort of circle with the master as center, and we moove along a* and b* axis, having a copy for each quadrant of the circle; we choose to not consider the L* axis, because it was already included in ΔE calculation - as seen before -, so the generated copies can be in circles with different L* with respect to the master. In the figure 9 we can see an example with a master chosen on the top right of the a*-b* circle: in the zoom - on the right figure - we can see the quadrants in which we are going to choose a Lab. The black circle has as center the master coordinates and as ray the maximal ΔE chosen, in this



(a) Circle around the master



(b) Zoom of the previous image

Figure 9: Representation of the circle from which we generate the copies

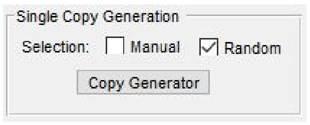
case 3. For each quadrant we generate at most 1000 copies, to have a copy that is in the selected range - [2-3] in our example - and with a non negative **RGB** value.

So we finally have n different, randomly chosen, masters, each with its four copies, generated in the selected range and one per a*b* circles'quadrant, showd in the simplified figure 9.

The generation of the copy isn't made with the white Gaussian noise because for each colour only a noise in specific ranges would have given specific results and in our case, having more random, different, colours, we couldn't choose a right noise for everyone. So we decided to use the criteria based on distance on the **L*a*b*** colur space, in order to decide to generate couples with smaller or bigger colour differences. The white Gaussian noise was instead helpful and useful in the case of a single copy, ase we are going to see.

After the generation of the multiple master-copy couples - that is made in background - these can be added to a TrainingSet or a TestSet, to be used later in a Neural Network. Multiple subsequently generation can be made, with different number of colours and different criterion.

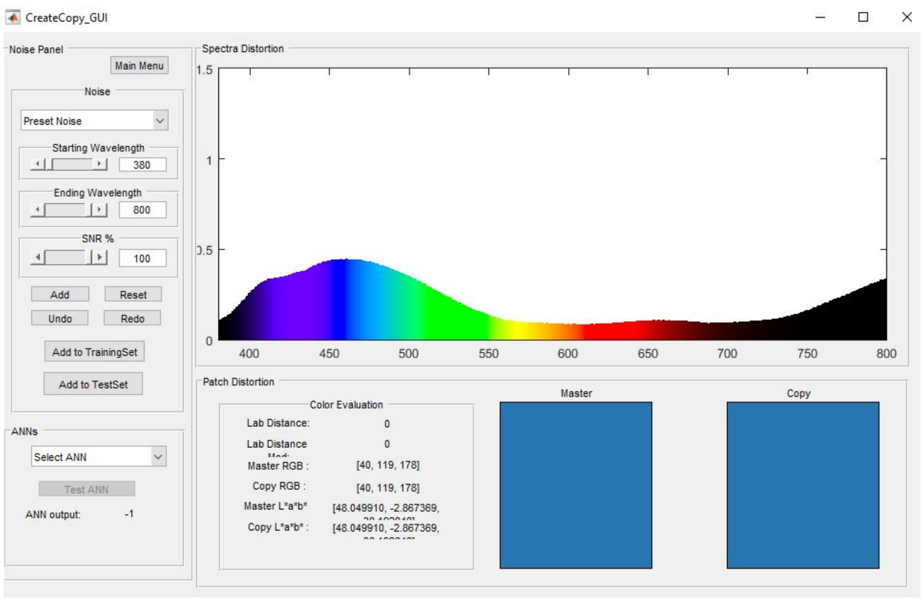
4.3 Single copy generation



Single Copy Generation
Selection: ☐ Manual ☑ Random
Copy Generator

Figure 10: Single copy generation Menu

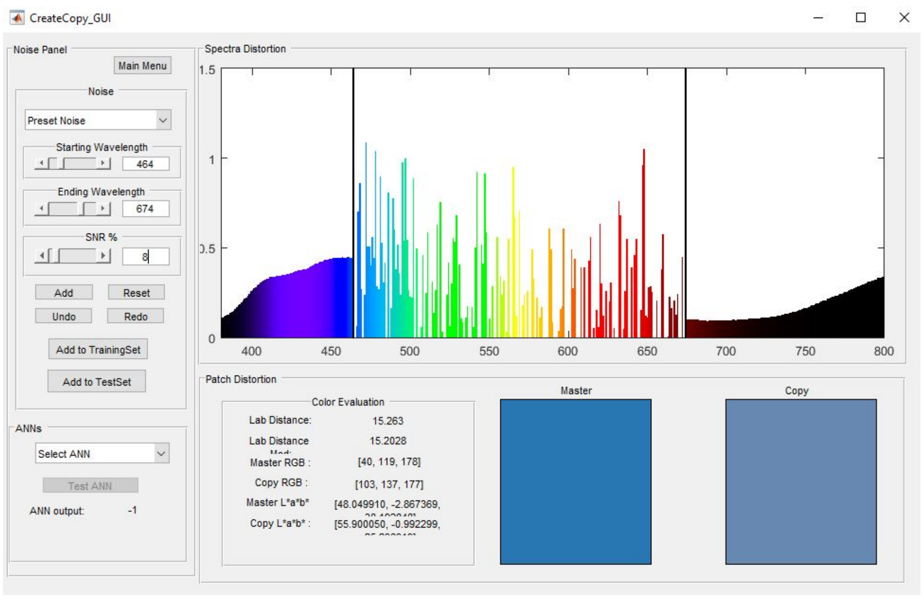
Single copy generation can be used to generate the copy of a single master, chosen between the ones in our dataset. Checking the respective checkbox, the master can be choose manually, from the image on the right, or random, by the tool. Clicking on "copy generator", another window is open.



Preset Noise
Starting Wavelength 380
Ending Wavelength 800
SNR % 100
ANN output: -1

Color Evaluation
Lab Distance: 0
Lab Distance Mod: 0
Master RGB : [40, 119, 178]
Copy RGB : [40, 119, 178]
Master L*a*b* : [48.049910, -2.867369,
Copy L*a*b* : [48.049910, -2.867369,

(a) Window to create copy



Preset Noise
Starting Wavelength 464
Ending Wavelength 674
SNR % 8
ANN output: -1

Color Evaluation
Lab Distance: 15.263
Lab Distance Mod: 15.2028
Master RGB : [40, 119, 178]
Copy RGB : [103, 137, 177]
Master L*a*b* : [48.049910, -2.867369,
Copy L*a*b* : [55.900050, -0.992299,

(b) Perturbated spectra

Figure 11: How to create a copy

In figure 11 we can see that, on the right side of the window, the spectrum of the chosen colour - initially not perturbated - is showed, and, below that, we can see, on the left, the specifics of master and copy - to be created - to understand their differences, whereas on the right their patches are shown. On the left side of the window a Noise panel is shown: clearly it can be used to choose a preset noise to apply to the colour - we are going to see how to create a preset noise -, or to apply a specific noise changing its frequency range and its signal to noise percentage ratio. In figure 11.b we can see that a noise in the interval [464, 674], with an SNR of 8 is applied, giving the spectra on the right side and the numerical results resumed in "colour evaluation" panel. Finally, the generated couple can be added to the TestSet or the TrainingSet or it can be tested with a neural network, using the dropdown menu in ANNs(*Artificial Neural Network*) panel. Clicking "Main Menu" button we can return to main menu visualization.

4.4 GUIs

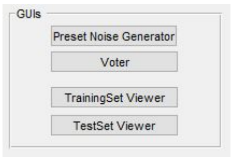


Figure 12: Gui Panel

The GUIs panel allow to access to four other interfaces: Preset Noise Generator interface, Voter interface, TrainingSet Viewer and TestSet Viewer. The first interface, as its called, represents a generator of white gaussian noise, useful to be stored and lather used to perturbate some chosen colours.

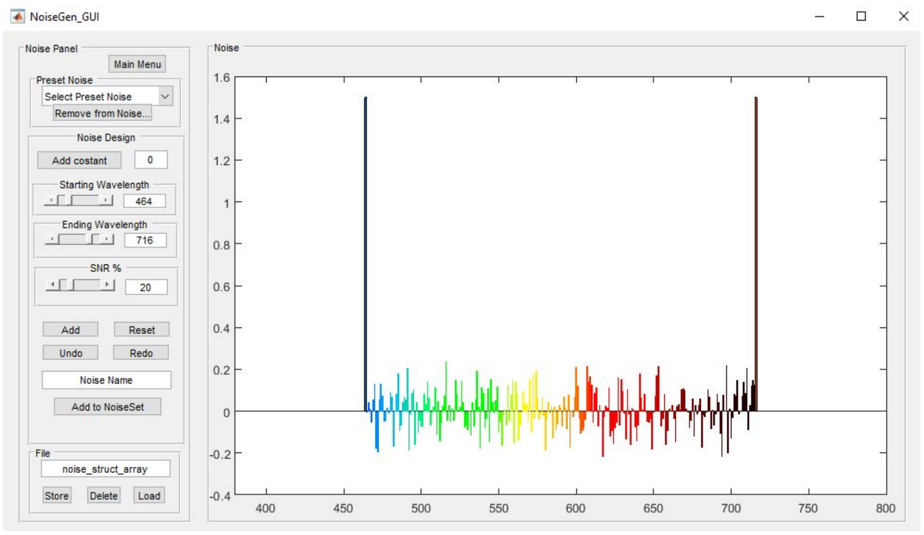


Figure 13: Noise Generator Interface

The window that is opened, when the button is pressed, is very similar to the one that allows to create a single copy, perturbing a master colour. As we can see in figure 13, it allow user to create a noise signal with the characteristics specified before, as the one created in the figure, with a range between 464 and 716 and an SNR% of 20. The noise can be added in a noiseSet to be used after in the "Single copy generator" to add a preset noise to a chosen master.

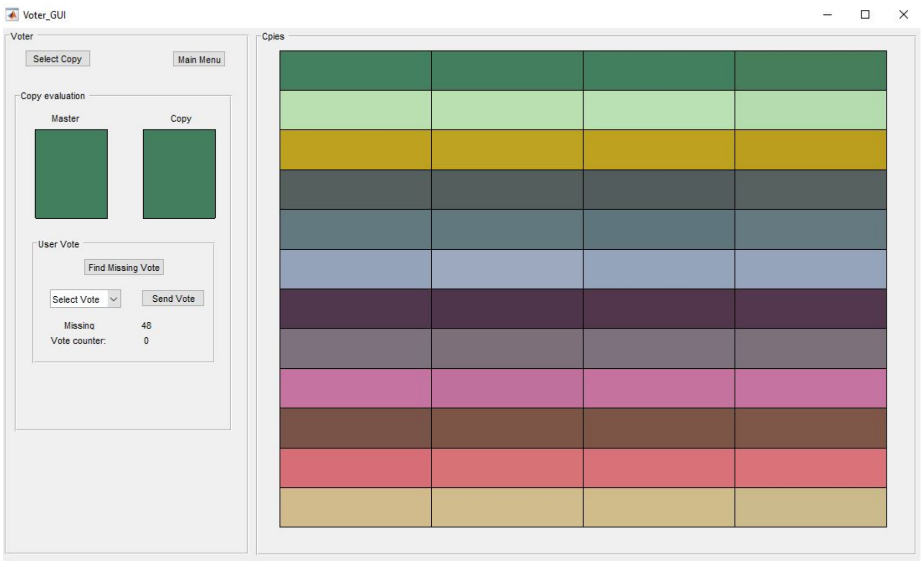
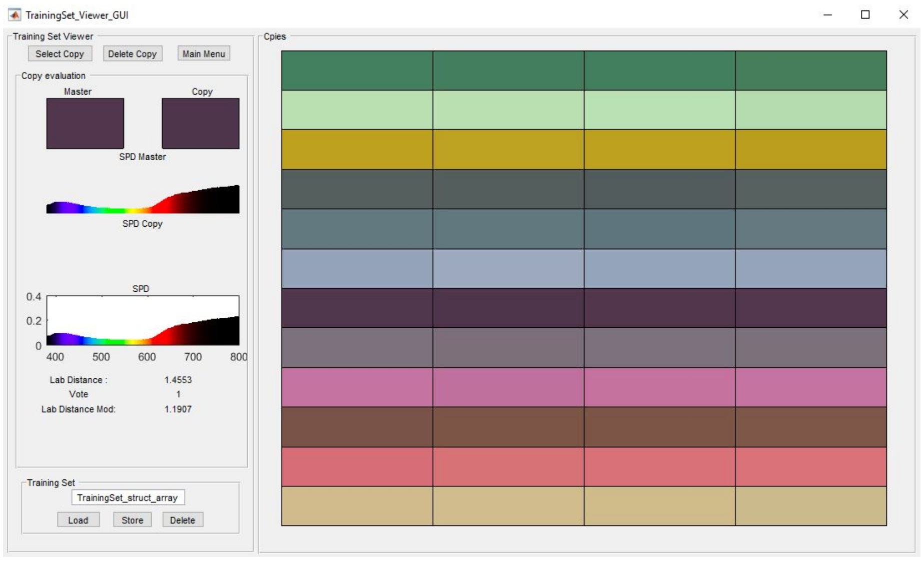


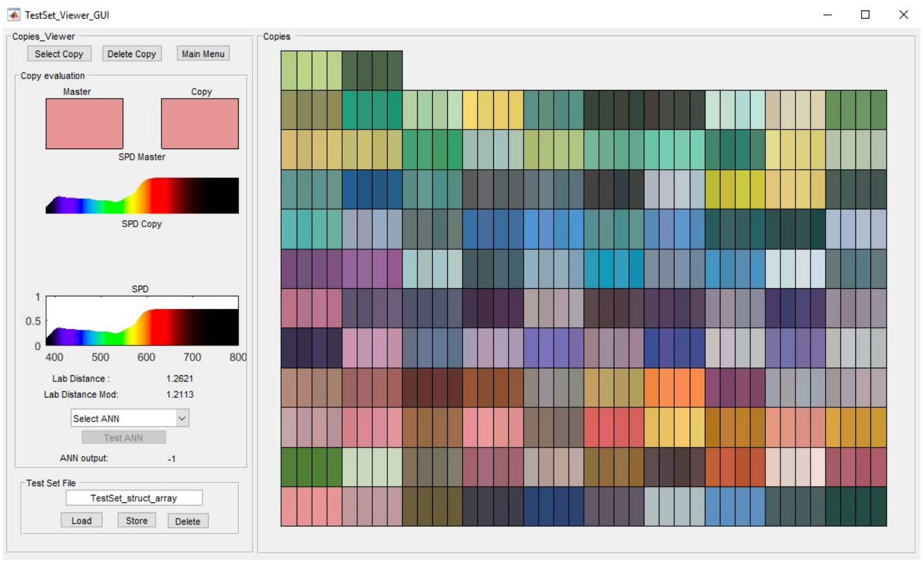
Figure 14: Voter Interface

Returning to the "Main Menu" we can click the second button "Voter", this opens a new window where the TrainingSet is showed on the right side - in figure 14 a voter with ten masters (40 copies) is showed-. In the left side of the window we can see a comparision between the first master and its first copy (the first colour patch at bottomleft of the right figure), and we can vote that using the dropdown menu. We can also select a specific copy by pressing on "Select Copy" or we can move on the pattern with the arrow keys. The vote for each couple can be chosen in the dropdown menu between: equal, almost equal, slightly different, different or detached. Each couple can be voted multiple times and a counter is available, to make user understand how many couples have been already voted. We are going to talk about *votes* and their role in our project in **Neural Network chapter**, when we'll talk about the generation of the different TrainingSets.



Lab Distance : 1.4553
Vote 1
Lab Distance Mod: 1.1907

(a) TrainingSet Viewer



Lab Distance : 1.2621
Lab Distance Mod: 1.2113
ANN output: -1

(b) TestSet Viewer

Figure 15: You can view the TrainingSet and TestSet generated

The last two windows that the user can open are "TrainingSet Viewer" and "TestSet Viewer" 15. These are useful to have a raffiguration of the generated colours, to be used in the neural network.

The TrainingSet viewer - figure (15.a) - shows the generated copies, four for each master, on the right side of the window - in this case we have ten masters, as before in the voter - and allows to select or delete a specific copy. On the left side of the window a comparison between the master and its selected copy is showed, with their patches, their spectra and lab distances values. If the copy wasn't generated, perturbating the master, the SPD of the copy is not showed.

Below this part, there is a panel that allows to load a pregenerated set or to store the actual one.

The TestSet viewer - figure (15.b) - is mainly the same as before, but shows the generated TestSet.

4.5 ANNs

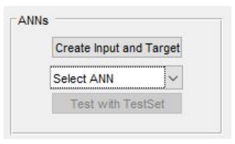


Figure 16: ANNs Panel

This panel allow to choose, with the dropdown menu, one of the Artificial Neural Network that have been trained - we are talking about them in next chapter - and test that with the created TestSet. The button "create Input and Target" allows to create input and target for the network, starting from the TrainingSet.

5 Neural Networks

In this part of the document we explain which are the neural networks we created and which choices has been made in order to have the best results. We studied multiple cases and developed different neural networks, with different neurons'number and different training sets.

We choosed to use **MLP** networks instead of **RBF** because, with some test we made, we saw that both the type of networks had good performances, with low **MSE**, but **MLP** gave better results with testset. In particular we saw that, with **RBF** we had a good training but we didn't have good generalization, so the network seemed to learn by the training set but didn't respond in a correct way to any test set. So we developed some *Multilayer Perceptron Networks* changing Training Sets and number of neurons.

5.1 MLP network with first Training Set

The first case we studied was a Multilayer Perceptron network with a training set including each master of the dataset.

5.1.1 Training Set

We generated the Training Set with our tool, generating for each master every kind of copy: very near (ΔE less than 2), near (ΔE between 2 and 3), a little far (ΔE between 3 and 5), far (ΔE between 5 and 10). So for each master of the dataset we had sixteen copies, four for each distance, having a TrainingSet with all the 1232 masters and 19712 copies. This allowed us to cover all the given dataset and have a uniform distribution over it, as we can see in figure 17.

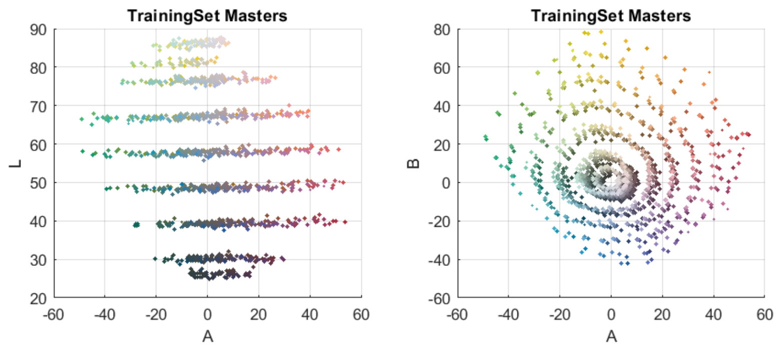


Figure 17: Training Set distribution for first network

So as **input** we had a 6×19712 matrix, where each column was a couple *master-copy*, the first three rows were the **L*a*b*** coordinates of the master colour and the other three were the coordinates of the copy. As **target** we had an array with 19712 entries, containing the ΔE modified by us, calculated for each couples.

5.1.2 Training

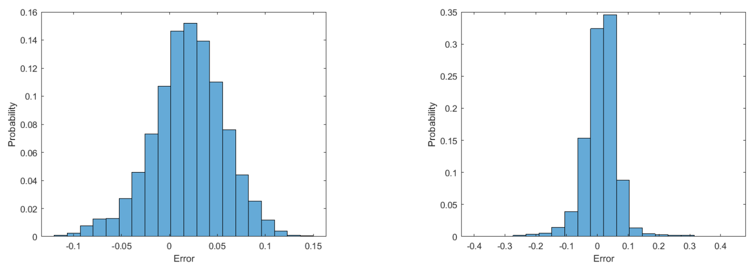
We trained different networks, with incremental number of neurons and different training function:

- we used Levenberg-Marquardt backpropagation training function (`trainlm`):
  - a network with twentyfive neurons;
  - a network with fifty neurons;
- then we used Bayesian regularization backpropagation training function (`trainbr`):
  - a network with twentyfive neurons;
  - a network with fifty neurons;
  - a network with a hundred neurons.

We noted that the networks trained with Levenberg-Marquardt training function reached a good result with fifty neurons and, incrementing the number of neurons, we had almost the same results, whereas with Bayesian training function we reached the best result with a hundred neurons. We could also see that the first networks had a computation time smaller than the second.

5.1.3 Test

To test the chosen networks - trainbr with a hundred neurons and trainlm with fifty neurons - we generated a testset with our tool, with the same characteristic of the training set: we took nine hundred couples uniformally distributed on the L*a*b* space, so with couples with a little ΔE, less than 2, larger, between 2 and 3 etc. We could see that the networks had good results, as underlined by the error probability distribution in figure 18, and that the output of the testset are a good approximation of the ΔE.



(a) Network with trainlm and 50 neurons

(b) Network with trainbr and 100 neurons

Figure 18: Error Probability Distribution

We could also generate a certain copy of a specific master and give that as input to the network, using the tool: a value less than **two** as output of the network means that the input colours can be considered equals. We made several tests and we saw that colour almost equals had a good output rate from the network [1].

5.1.4 Results

Finally, with the training set including all the masters with copies at all distances we had two good networks with different number of neurons and training functions. Both of them gave a good approximation of the Lab distance we computed as said before. Obviously the human perception was included in this calculation and, considering the Training Set without votes we only have an approximation of the Lab distance modified, that, however, gave a right *equality rate* for almost every master-copy couple.

5.2 MLP network with reduced Training Set

The second case we studied was a Multilayer Perceptron neural network with a training set including each master of the dataset but not considering copies at each distance.

5.2.1 Training Set

We generated the Training Set with our tool, generating for each master only near copies, that are the copies that usually are not easily recognisable by human eye. So we generated eight copies for each master: four near (ΔE between 2 and 3) and four less near (ΔE between 3 and 5). Here, in figure 19 we can see the distribution of these training set, that is however uniform on our dataset.

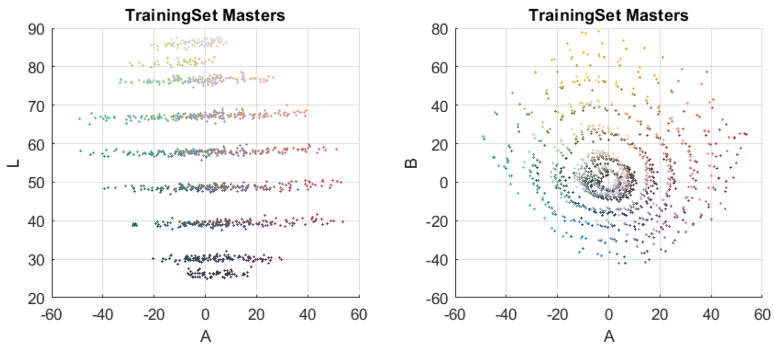
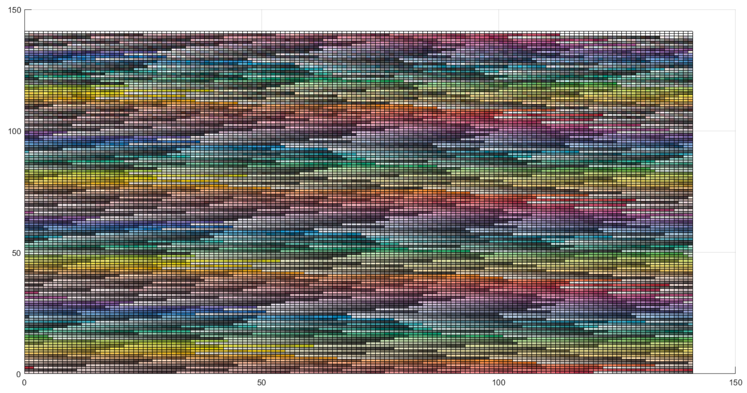


Figure 19: Training Set distribution for second network

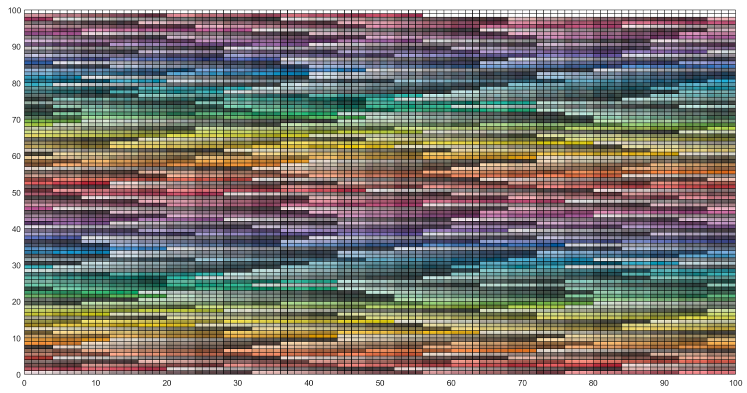
So these TrainingSet was smaller than before, with all the 1232 masters and 9856 copies. So as **input**, we had a 6×9856 matrix, composed as before and as **target** we had an array with 9856 entries, containing the ΔE modified by us, calculated for each couples.

We can also see the differences between this training set and the one before by comparing the two images in figure 20, that represent the training set as side-by-side patches.

[1]The network to be selected to make the test is "All_Masters_0.5to2_2to3...."



(a) first Training Set



(b) reduced Training Set

Figure 20: First image is the first training set, that is larger than the second

5.2.2 Training

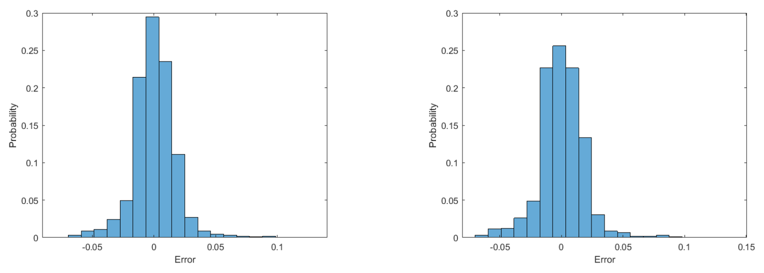
We trained different networks, with incremental number of neurons and different training functions:

- we used Levenberg-Marquardt backpropagation training function (`trainlm`):
  - a network with fifty neurons;
  - a network with a hundred neurons.
- then we used Bayesian regularization backpropagation training function (`trainbr`):
  - a network with fifty neurons;
  - a network with a hundred neurons.

Having a smaller training set than before, we needed a smaller computation time to train the different networks and we could also try networks with more neurons, even if with `trainlm` function the network with fifty neurons remains the best, with the following ones that obtained the same results.

5.2.3 Test

We tested, as before, the network with trainlm and fifty neurons and the one with trainbr and a hundred neurons, generating a testset with our tool with the same characteristics of the training set: we took only couples with a little ΔE, between 2 and 5. We saw by the test that the two network are a good approximation of lab distance mode and gave as output a good *equality rate*. The main difference with respect to the networks before can be seen for the cases not included in training set: in fact in this case, if we test a network with a couple with a ΔE larger than 5 or smaller than 2 the network will give a wrong output. This is due to the fact that the network doesn't know couples with that ΔE, so it can't understand these kinds of difference. We can also see the error probability distribution in 21.



(a) Network with trainlm and 50 neurons

(b) Network with trainbr and 100 neurons

Figure 21: Error Probability Distribution

Also in this case graphical tests can be made with MATLAB tool [2].

5.2.4 Results

Also in this case, as before, we choose two network that have the best performances: in each case we had a good approximation of lab distance, so a good output with respect to couple of similar colours. As said before, the problem of these network is that they can't recognize very different colours, but this is not a problem if we consider that the networks have the purpose to understand similarity of colour, not easily distinguishible by humans. So it is not important that networks understand if two colours are very different or equals, but it's important that it distinguishes colours that are similar, so with a ΔE value between 2 and 5.

[2]The network to be selected to make the test is "All_Masters_2to3_3to5"

5.3 MLP network with first voted Training Set

In the cases before we used training sets where colours were not voted, so as targets we only have the ΔE values, calculated with our modification. The networks seen before gave an approximation of ΔE with perception that was introduced as said in third chapter. So we decided to introduce votes, as already described, to have more precise Training Sets.

5.3.1 Training Set

The first Training Set we built contained 2000 couples of near colours, with ΔE between 2 and 3 and 800 couples with a ΔE between 3 and 5. Even in this case we took only similar couples, as in the case before. Obviously we had a different distribution with respect to the cases before: as we can see in figure 22, colours are distributed in an uniform way over the space, but also in a more "randomic" way.

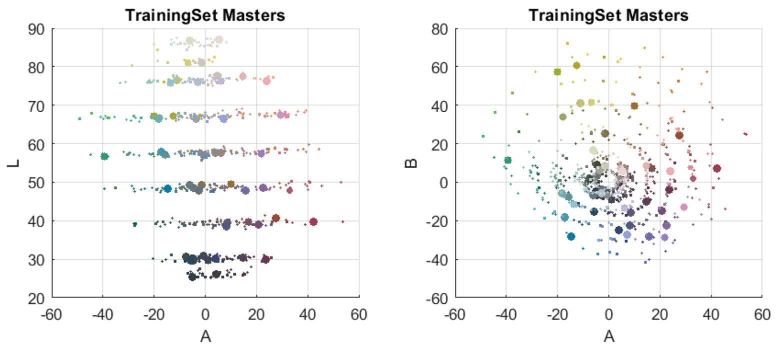


Figure 22: Voted Training Set distribution

After the generation of these couples, we used the Voter interface to vote them. We separately - with different monitors - voted each couple, with a very long and boring work and then, for each couple, we calculated the average between the votes. Each vote represents a real number that is multiplied to ΔE when the input and target are created starting from the TrainingSet. In particular a vote *equal* is a rate of 0.5, that halves the ΔE value, a vote *almost equal* represents a 0.8, that again decrease the value, whereas the other votes are used to increase ΔE. Details of these calculation are available in file *Voter_ GUI.m*, where all the callbacks functions are built. So we finally had 2800 input with 2800 targets that are *equality rates* based on ΔE and our votes.

5.3.2 Training

As before we built MLP with `trainlm` and `trainbr` training functions:

- we used Levenberg-Marquardt backpropagation training function (`trainlm`):
  - a network with fifty neurons.
  - a network with a hundred neurons;

- then we used Bayesian regularization backpropagation training function (`trainbr`):
  - a network with fifty neurons;
  - a network with a hundred neurons.

In this case we had better performances with trainbr function, so we discarded the networks made with trainlm and we used only the one maid with trainbr and a hundred neurons.

5.3.3 Test

In this case we can test the network but we need a voted test set, because obviously, another generated testset will not have the same target given as output of the network. Using the tool we can test the precision of the network, that has very good performances with similar colours. As in the case before it has some problems with more distant colours, that however are not so important for our work [3].

5.3.4 Results

The network trained with trainbr and with a hundred neurons is very good to analyze colour differences because it doesn't only approximate the ΔE but also the human perception included in the Training Set with the votes

5.4 MLP network with complete and voted Training Set

In this case we had a sort of merge of the training set seen before and we built two MLP, one with `trainlm` and a hundred neurons, and one with `trainbr` and a hundred neurons.

5.4.1 Training Set

As said before, in this case we merged the Training Sets used before: we took the 2800 couples voted and we add to them som other randomly generated couples. We generated 500 couples with ΔE between 0.5 and 2, 500 with ΔE between 2 and 3, 500 with ΔE between 3 and 5 and 500 with ΔE between 5 and 10, obtaining 8000 couples. These couples weren't all good so we discarded some of them: in fact we had 1549 couples similar to the voted ones. In figure 23 we can see the distribution of this training set, that, as before, uniformally covers the dataset space.

[3]The network to be selected to make the test is "N2000_M800_A208_Voted"

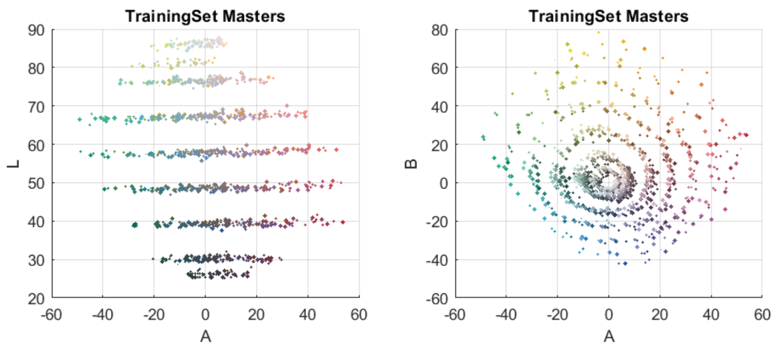


Figure 23: Voted Training Set, merged with firs Training Set, distribution

So we finally had 6541 not voted couples and 2800 voted, so the 30,27% of couples were voted, and we had a good distribution on the training set. The differences between this training set and the one before is obviously in the target, because in this case we have some couples that are not voted, so with an *equality rate* not changed with the votes.

5.4.2 Training

As said we only trained two networks because we expected to have the best results with a hundred neurons, as before. In fact the network with best performances is the one trained with trainbr function, as seen before for the case with voted training set.

5.4.3 Test

In this case we have voted couples and not voted couples, so we expected that our network could approximate even not voted test set. In fact we tested the network with the testset that covers the whole space - ΔE between 0.5 and 2, between 2 and 3 etc.- and we saw that the given results were a good enhancement of the lab distance mode, obtained thanks to our votes[4].

5.4.4 Results

Finally we can say that this network is the best for our project because it is not only based on the voted set, so it can recognize also colours not voted and so it can cover the whole space, as for example cases with very distant colours.

[4]The network to be selected to make the test is "TopVoted""

6 Conclusions

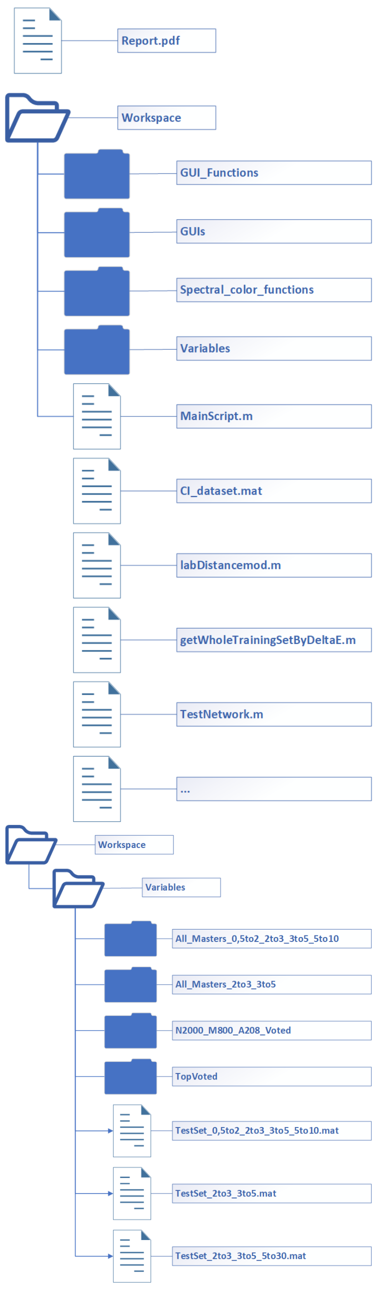
In this document we saw which problem we had to fight and which was the purpose of our project. We made an introduction about colours and perception, giving our solution to some of the problems of ΔE and introducing our ΔE modified. Then we showed the MATLAB tool we developed, in order to work on colours with a simple interface and finally we talk about the neural network we built. For each neural network we explained which Training Set was used and which training algorithms, underlining which networks gave the best results for each training set.

Finally we can say that the last neural network, an MLP with training function `trainbr` and a hundred neurons, trained with the Training Set composed by voted and non-voted couples, reached the best results in recognizing colours. Our tool allows to utilize this network - and all the others - and make test on colours couples, with an *equality rate* that is returned by the network.

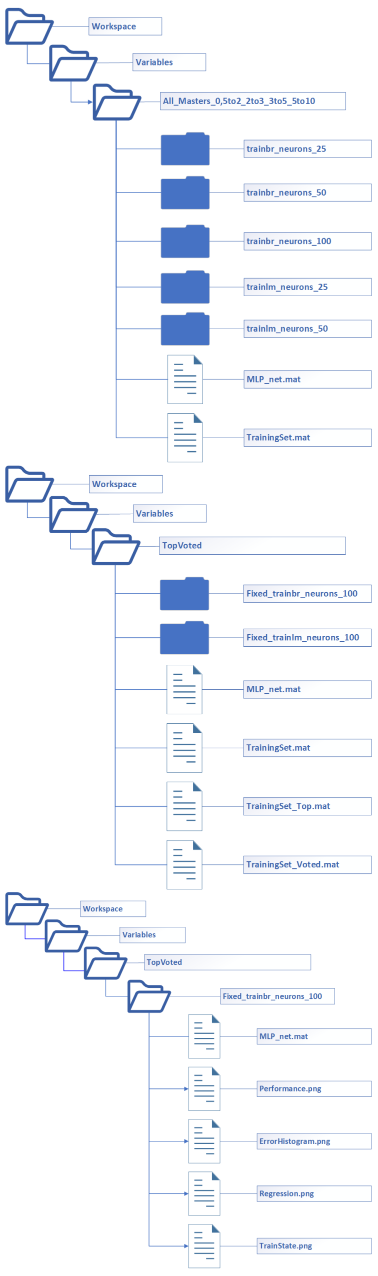
Appendices

A Tools Documentation

A.1 Directories and Scripts organization



The "Workspace" directory is organized in many folders, to simplify the usage and inspection of tool's files. In the figure on the left we can see the organization of the main directory, where is placed the script `MainScript.m`, that automatically includes the subfolders needed by the tool.
The **scripts** contained in this folder are related to the calculation of colors starting from the spectrum, the computation of L*a*b* Distance and *Lab Distance modified*, the perturbation of spectra using white Gaussian noises, the *creation of the copies* and finally the *building and test of the networks*.
In order to improve the portability of the tool, we have included, in "spectral_color_functions" folder, the scripts ***Spectral and XYZ Color Functions***, available in Matlab color toolbox.
In "GUIs" directory you can find the interfaces described before, organized in one .m and one .fig file for each GUI. The execution of `MainScript.m` will start automatically the Main GUI and from that all the others are reachable.
In "GUI_Functions" folder we included the scripts used by the **GUIs**, to plot and manage graphical stuff as patches'plots, spectras'drawing or masters distribution in the L*a*b* space.
The "Variables" directory finally is used to store variables and is the main folder of the "*store*" and "*load*" options available in the GUIs; in this directory there are all the cases we studied, and in the main directory there are also some TestSet used to test the networks.



Each directory includes the files for one of the neural networks we developed: the **TrainingSet** used to train the network is stored in `TrainingSet.mat` file as *TrainingSet_struct_array* whereas the best **trained network** variable is contained in *MLP_net.mlp*; for each configuration of the network there is a dedicated subdirectory that contains the network itself and some pictures stored after the training.
Here we show two directories: the the first one is the directory of the first network and the second is the directory of the last network we built. In the first directory (All_Masters_0.5to2_...) we have five different configurations of the network, corresponding to five different subdirectories which name specifies the training algorithm used and the number of neurons (for some cases the training algorithm name is missing, this means that the network is trained with trainlm);
In the second case two folders labeled with **Fixed** are present, this means that the TrainingSet used is, or is composed, by the *N2000_M800_A208_Voted* TrainingSet - the one voted - but fixed, removing the **208 votes** of the patches with a distance greater then thirty.
For the **TopVoted** case, contained in "TopVoted" folder, the TrainingSet is created by merging the two training set contained in files `TrainingSet_Top.mat` and `TrainingSet_Voted.mat`, and is contained into the `TrainingSet.mat` file, as in the other cases.

A.2 Networks Testing without using the GUI

It is possible, for a user, to test the networks also without using the developed tool. The first way is to import in MATLAB the file `MLP_net.mat` of the chosen network, this will import the variables *net* and *tr*, returned after the training, and that can be used to test the network. You can use as input the **L*a*b* coordinates** of a master cuncatenated with the **L*a*b*** coordinates of its copy in a column vector 6×1. Another way is to test the chosen network by executing the function, present in the "workspace" folder, *TestNetwork(who, input)* that takes as input the values associated to each network, reported in Table1, and the 6×1 column vector of input and returns the *equality rate* given by the network.

Network Name	**Who**
All_Masters_0,5to2_2to3_3to5_5to10	1
All_Masters_2to3_3to5	2
N2000_M800_A208_Voted	3
TopVoted	4

Table 1: Networks

In the situation above we need to use the l*a*b* coordinates instead of the colours'spectra, that however can be used and converted to L*a*b* using the function *convertSpectrum(spectrum)*.

A.3 Struct used in TrainingSet and TestSet

In "Variable" folder we can see the TestSets and TrainingSets created using the GUI - every set created in the GUI is stored here -. The *struct* used for this purpose is reported in Table2; this same structure is manually extendible, but obviously a reload of the GUI is needed, if it is already open.

Field Name	**Field Type**
spd_copy	421x1 double
spd_master	421x1 double
DeltaE	double
DeltaE_mod	double
master_rgb	3x1 double
copy_rgb	3x1 double
master_lab	3x1 double
copy_lab	3x1 double
vote	double
count_votes	double[5]
sum_votes	double

Table 2: Base structure

The fields *DeltaE* and *DeltaE_mod* can be computed using the functions *labDistance(lab1, lab2)* and *labDistancemod(lab1, lab2)*, the *master_lab* and

[5]By default, MATLAB®. stores all numeric variables as double-precision floating-point values

copy_ lab can be computed starting from the spectrum with the function *convertSpectrum(spectrum).* The default *vote* must be 1, instead *count_ votes* and *sum_ votes* must be 0, also the default Spectrum for the *spd_ copy* field must be a column vector 421×1 of zeros, but can be used a different one coherent with the fields *DeltaE*, *DeltaE_ mod*, *copy_ rgb* and *copy_ lab*.