Simulating ICC Color Profiles Using Regression Techniques Based on Machine Learning

Jordi González Cano

Resum– La gestió del color (CM) és el procés que permet transformar els colors d'una imatge en un espai de color a un espai de color diferent utilitzant els perfils de color ICC. Tot i que els perfils ICC són molt utilitzats a l'hora de fer CM en tot tipus de dispositius com ara pantalles i impressores, una de les limitacions és la dificultat de la paral·lelització de les transformacions i així poder incrementar la velocitat a l'hora de dur a terme les transformacions de colors en imatges grans, com ara les imatges d'impressió. Les Xarxes Neuronals pot solucionar aquesta limitació i fer les transformacions de color molt més ràpides. En aquest projecte s'ha dut a terme una recerca amb diferents aproximacions d'ús de Xarxes Neuronals per tal de poder realitzar transformacions de color i comparar els resultats quantitativament i qualitativament amb els obtinguts usant els tradicionals perfils ICC. Els models finals obtinguts mostren que és possible simular els perfils ICC amb les Xarxes Neuronals i que poden anar fins a 15 vegades més ràpid que les transformacions realitzades amb els perfils ICC quan les Xarxes Neuronals corren sobre GPU.

Paraules clau– Gestió del color, perfils ICC, Aprenentatge Computacional, Regressió, Transformacions de Color, Xarxes Neuronals, Little CMS, paral·lelització amb GPU .

Abstract– Color Management (CM) is the process that allows to transform image colors from one color space to a different color space using ICC color profiles. Although ICC profiles are widely used for CM in all sort of devices such as screens and printers, one limitation is its difficulty to be parallelized and be able to increase speed in performing color transformation in large images as those in printing. Neural Networks can solve this limitation and make color transformations much faster. In this project research was carried out on different approaches for using Neural Network into creating color transformation and comparing quantitative and qualitative results to those of traditional ICC Profiles. The final models obtained show that it is possible to simulate ICC Profiles by means of Neural Networks and that they can perform up to 15 times faster than the transformations carried out with the ICC Profiles when NN run on a GPU.

Keywords– Color Management, ICC Profiles, Machine Learning, Regression, Color transformation, Neural Networks, Little CMS, GPU parallelization.

1 Introduction

HAVE you ever thought why an image seen from different devices, such as different kind of screens or paper, look all quite similar? This is carried out by means of the *Color Management* (CM), which computes a transformation of the colors of an image that allows to see them as similar as possible in two different devices.

CM uses the *International Color Consortium Profiles* (ICC Profiles) to perform these transforms. These profiles are associated with one device and they contain all the information needed to accomplish the transformation. The ICC profiles contain an input color space and an output color space, called *Profile Connection Space* (PCS). The PCS is the common color space through which two arbitrary color spaces can be connected and allows color transformation to be colorimetric constant. This is how the colors in an image

• Contact e-mail: jordigc2@gmail.com
• Specialization done: Computation
• Work tutored by: Fernando Vilariño (Dep. Ciències de la Computació)
• Work tutored by: Jaume Vergés (HP Inc.)
• Year 2017/18

can be seen similar in two different devices. To compute the CM is used the open source library *Little Color Management System* (LCMS)[10], but there are other software.

To implement the transformations, an ICC profile contains a *Look Up Table* (LUT) which provides the transformed color from an input color directly. The problem is that, for example, in RGB there are three dimensions of colors and each one can have a value from 0 to 255, giving as a result 16.5 millions of colors, this is a large amount of colors. What is needed is an ICC profile that does not use too much memory and to have 16.5 millions of inputs is not the solution. A discretization of each color coordinate in 17 points is used to minimize the size of the LUTs, giving as result $4,913$ points, inputs in the LUT, that allows to compute all colors. From these colors, any intermediate color can be interpolated by doing a multivariate/trilinear interpolation algorithm. Now the profiles are relatively light [13].

This structure has two limitations. On the one hand, the interpolation is a tedious process that requires a lot of time and effort and on the other hand, ICC profiles are a rigid structure difficult to implement in a GPU to parallelize and speed up the computation.

In this work we propose to use *Neural Networks* (NN) based on multivariate regression to perform the color transformation computation. We propose such structures because they can simplify the process of profile creation and allows the usage of GPU's in order to parallelize the computations. In Section 2 it will be explained the objectives of the project. In Section 3 it will be explained the state of the art. Then in Section 4 it is explained all the methodology. Then in Section 5 and 6 the results are showed. Finally, in Section 7 there are the conclusions of the project.

2 OBJECTIVES

The purpose of this project is to create a model to perform color transformations more accurately and faster than current ICC Profiles. Several objectives have been proposed to achieve this in this project. There are two type of objectives:

- The following objectives are to implement the different software for the project:
  1. Learning, installing, and running color transformation using LCMS: This is the first and the most important to complete, because is the reference for the learning.
  2. Learning, installing, and running different approaches to perform multivariate regression which are the following:
     - Linear and polynomial Regression (LR and PR)
     - Regression using Support Vector Regression (r-SVR)
     - Regression using the Neural Network from the state of the art
     - Regression using an improved Neural Networks
  3. Implementing an alternative framework that allows the training, validation and testing of different models, optimizes the learning parameters, and keep track of the results. We want to know how each model behaves in terms of:
     - Accuracy: This can be measured in terms of the Euclidean distance or to be more precise in terms of the ΔE. The ΔE is a way of calculating the distance of two colors, due to the color space is not uniform and the distance scale is different depending on the color value. The smaller the distance the better the color generated with the model will be.
     - Time: Compare all the models how much time they take to transform different images and over different machines, such as CPU or GPU's.
     - Size: Use the smallest model with the highest precision and with the less number of colors used to train.
  4. Implementing an alternative framework for running experiments and obtaining results where we can enter an image and obtain a new one in another color space with the models trained and compare it with the one transformed with the ICC Profiles.
- Experimental Objectives: The objectives once a functional NN has been obtained are:
  1. Obtain a model from a small representation of the color space, input and output.
  2. Verify in a device that the results obtained are perceptually what it has been expected.
  3. Confirm that the NN is the best architecture, in terms of accuracy, for this problem and not the Regression models and neither the ICC-Profiles.
  4. Measure the time performance of NN vs ICC Profiles for color transformation
  5. Check if the model using parallelization is faster than the one that is not using GPU.
  6. Do a more accurate approximation to the ICC-Profiles using NN. This means that instead of having one model that goes from the input color space to the output, two models are used, going from the input color space to the PCS(Lab color space) and from there to the output color space.

3 STATE OF THE ART

ICC profiles are a standard created in the 1990s by the *International Color Consortium* (ICC), which is formed by a group of big companies. The usage of the ICC profiles has increased notably every year, due to the appearing of all kind of devices to manage colors, such as screens, cameras and printers.

In the last decade there has been a lot of research to improve the color management with NN. Results show that the accuracy of NN is better than any regression method as can be seen in the papers [4], [5] and [6].

The paper by Baohui Xu, Yan Li and Jie Zhang [4] compares several *Back Propagation Neural Network* (BPNN) architectures to other data structures. The difference between such BPNN architectures is that they do not have the same number of nodes per layer and they have different number of hidden layers. Despite the input and the output layer have all 3 nodes, the hidden layers vary from 8 to 20 nodes. The number of hidden layers goes from 1 to 3 hidden layers.

BPNN modifies the weight depending on the error produced in the output from a certain input and always tries to find the minimum error. The error produced with the BPNN model is calculated with the euclidean distance between the predicted color and the expected one with the following equation.

$$\Delta E_{(Lab1,Lab2)} = \sqrt{(L_1 - L_2)^2 + (a_1 - a_2)^2 + (b_2 - b_2)^2} \tag{1}$$

Paper [4] is focused in finding the BPNN architecture that produces the smallest error. They say that adding more hidden layers was not a synonym of obtaining a smallest error in the test. The same happened when fixing the number of hidden layers and incrementing the number of nodes in each layer. So, the conclusions of this paper are that BPNN allows parallelizing, speeding up the process and that the biggest BPNN generated the smallest error. However, for a light model, finding the smallest BPNN with the least error is the goal.

The paper by Zhi Chuan and Zhou Shi-Sheng [5] mentions that 3D-LUT are perfectly accurate if we can have a large amount of colors in them. Nonetheless they are remarkable heavy in weight and the matching will be therefore slower. Also if a lighter 3D-LUT is created it will not be able to represent the essential colors. For such reasons, a solution with BPNN has to be implemented to improve the speed and the accuracy. The BPNN implemented in paper[5] has 3 hidden layers and an input and output of 3 neurons. It is studied the influence of the number of neurons in the hidden layer. It states that the larger the amount of neurons in each hidden layer the more accurate the approach is, but such architecture can easily generate overfitting. This means that it does not generalize, giving as a result high errors with the validation samples. On the other hand, if the hidden layers have a lower number of neurons they will not fit the data and even the training and the testing samples will give high errors. Finally, the architecture used in the BPNN has a *logistic-sigmoid* (log-sig) activation function in all three hidden layers and a *pure linear* (purlin) activation function in the output layer. The log-sig function is used when we want an output between the range of 0 and 1 and the purlin is recommended in the output layer of a NN.

The paper by Cao Congjun and Sun Jing [6] states that a BPNN will improve the accuracy of the color management. However, it says that the main problem with BPNN is its a low speed to converge while training and gets stucked in the local minimums of the error function. Therefore, the solution proposed in the paper is to use a *Generalized Regression Neural Network* (GRNN). *"GRNN has very strong nonlinear mapping ability, flexible net structure and high fault tolerance and robustness"*, as explained in paper [6].

The GRNN structure implemented in the paper has one hidden layer and uses a *Radial Basis Function* (RBF). A RBF is a Gaussian that if looked from above it is a circle where the center is the mean and the radius is the deviation of the data inside the Gaussian. A point closer to the center of the Gaussian will produce a larger output value than a farther point from the center. Finally, there is a Linear function in the output layer. The expected color is obtained using an image software, and the steps done are:

1. The expected color and the GRNN output color, not Lab, are transformed to Lab with the image software.
2. Now the distance can be calculated with the Equ. 1.

4 METHODOLOGY

In this section the progress done to obtain the final models to perform the CM is explained.

4.1 Machine Learning Tools Comparison

The idea of the project is to improve the NN approaches explained in the previous section. However, first we want to make sure that the transformation can not be done with classical regression models, using the SkLearn library [1], such as:

- Linear regression.
- Radial Basis Function (RBF) regression.
- Polynomial regression.

These models were compared with the Neural Network in papers [4, 5, 6], which has the following structure:

- Three hidden layers.
- 20 neurons in each hidden layer.
- The hidden layers have a *sigmoid* activation function and the output layer has a linear activation function.

The data set used to train and test all the models has 130K *Adobe RGB* colors. These colors are transformed using the LCMS and the combination of two profiles, the input and the output profile. Giving as a result the output colors in another color space, such as *Apple RGB* or CMYK, among others. In this project it has been chosen to compute the CM from *Adobe RGB* to *Apple RGB* because nowadays the new printers are starting to work with RGB instead of working in CMYK.

4.2 The Neural Network Architecture

Taking as reference the NN from the state of the art, some combinations of layers and neurons were tested to find which one better fitted the data set. Four different type of structures were tested:

- **Square Network**: All the hidden layers had the same number of neurons. The tests done were changing the number of neurons and the number of layers. For more details see Fig. 13 in appendix A.1.

- **Increasing Network**: Each hidden layer had more neurons than the previous layer. In these tests the maximum number of neurons was specified and in each layer the number of neurons increase until reaching the maximum specified. For more details see Fig. 14 in appendix A.1.

- **Decreasing Network**: Each hidden layer had less neurons than the previous layer. The maximum number of layers was specified in the tests, but now it started with the maximum number of neurons and it decreased in each layer until reaching to 1 neuron. For more details see Fig. 15 in appendix A.1.

- **Inc-Dec Network**: The number of neurons increased in each layer until a maximum number of neurons and then started to decrease in each layer until a minimum number of layers. In this structure, it was also specified the maximum number of neurons and first in each layer it increases until reaching the maximum and then decreased until reaching one neurone in the last hidden layer. For more details see Fig. 16 in appendix A.1.

The next step was to proceed to identify which is the best combination of activation functions.

The first test modifying the activation functions was performed with the activation functions of the NN created from the papers [4], [5] and [6]. This NN contains a *Sigmoid* activation function, which can be seen more details in Fig. 17 from appendix A.1. The problem with this function is that it can make a NN get stuck in the training. On the one hand, this is why the results are not as good as expected. On the other hand, the output activation function is lineal. This means that what goes in the function returns a value from $[0...255]$, but the output may be out of the limits of the interval $[0...255]$. Consequently, some outcomes are far from the range of colors. For more details see Fig. 18 in appendix A.1.

The second test was performed with a *tanh* activation function in the hidden layers and a *Leaky ReLu* in the output layer. The *tanh* activation function is similar to the *sigmoid*, but with the difference that the *tanh* activation function is in the range 1 to -1, which allows to have a stronger mapping of negative and zero values. As a consequence, it is less likely that the training process gets stuck. The representation of the tanh activation function can be seen for more details in Fig. 19 of appendix A.1.

The *Leaky ReLu* activation function is similar to the linear activation function, but this function puts all the negatives outputs between 0 and a small negative number depending on a σ. This σ prevents the function from saturating, which would stop the training process, as it happens with the *ReLu* activation function. The positive values are then calculated as if they were on a linear function. This again might create the previous problem of having output values out of $[0...255]$ range. For more details Fig.20 from appendix A.1 shows a representation of the Leaky ReLu and the ReLu activation functions.

4.3 Tunning the Data Set

To figure out whether there can be any improvement, the next tests were performed on tunning the dataset. The data in the previous tests are created using LCMS and no color was removed or modified before the training. The problem with this data set is that there can be some colors that have values over 255, which are impossible to represent in an 8 bit image. A color over 255 or below 0 goes to the other side of the range when is represented in an image. This is caused because the *Adobe RGB* has a wider gamut than the *Apple RGB*, as it can be seen in Fig 21 from appendix A.1 for more details. Therefore, if the *LCMS* transforms a color from the Adobe profile that is out of gamut in the Apple profile, the resulting color in the output of the LCMS will have at least one dimension value over 255. To solve this, two options were tested before training the model:

1. Modify the output colors by substituting with the value 255 all the values over 255.

2. Remove the out-of-gamut colors when creating the data set.

The next tests are done by fixing two of the three dimensions and increasing the third dimension from 0 to 255. This will be useful to see how the model fits the different data set used.

To detect if a color is out of gamut in the output profile the following steps are carried out: (Steps are illustrated in Fig. 1)

1. Go from the input color space to Lab color space using the input color profile, obtaining the colorimetric *Lab1* value.

2. Transform *Lab1* to the output color space with the output color profile creating the *OutColor1* color.

3. Transform the *OutColor1* color back to Lab colorspace using the output color profile creating the *Lab2* color.

4. Transform *Lab2* to the output color space with the output color profile creating the *OutColor2* color.

5. Transform *OutColor2* back to Lab color space using the output color profile creating the *Lab3* color.

6. $Diff1 = DeltaE(Lab1, Lab2)$ and $Diff2 = DeltaE(Lab2, Lab3)$.

7. If *Diff1* and *Diff2* are below a threshold(thr=5) will mean that the input color is **inside gamut**.

8. If *Diff1* is over the threshold and *Diff2* below the threshold will mean that the input color is **out of gamut**.

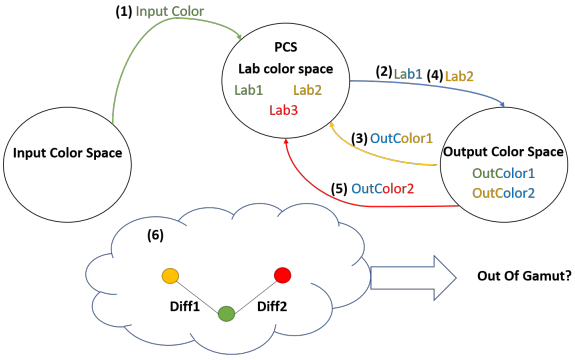


Fig. 1: Steps to verify if a color is Out Of Gamut

4.4 Two Steps Color Management

After seeing that is possible to carry out the Color Management directly from the input color space to the output color space, and with no high errors, in this section we explain a second approach. This approach consists on computing the same steps as in the CM with the ICC-Profiles, that are:

1. From the input color space of the first ICC-profile to PCS, which is in the Lab color space.
2. From PCS to the output color space of the second ICC-profile.

This is possible due to the fact that one profile can go from the input color space to the PCS, which is the colorimetric color space and common for all the ICC-Profiles, used to make the connection between two ICC-Profiles. The ICC-Profiles are bidirectional, which means that can transform from the input color space to the PCS or from the PCS to the input color space. This allows a large amount of combinations between ICC-Profiles through the PCS.

To make this approach with NN, four models are needed to go from an input color space to an output color space and in the inverse way. To prove that this approach is possible, three models were trained:

1. From *Adobe RGB* color space to Lab color space.
2. From Lab color space to *Apple RGB* color space.
3. From Lab color space to a *Device RGB* color space. This printer profile corresponds to a glossy paper using a Jaguar printer from HP Inc.

With this three models the CM can be done from *Adobe RGB* to *Apple RGB* or to *Device RGB* but not to the other way round, which will need 3 more models to make it possible. All the three models have 2 layers of 60 neurons using a *tanh* as activation function in the hidden layers and a *Leaky Relu* at the output layer.

The main problem of this approach is that the models trained are not perfect, they have errors, so if two models are combined this error will be added. To prevent this the models have to overfit the dataset, as much as possible, to obtain a lower error. The overfitting would be also useful in the case the models were trained with the whole set of colors possible. However, this is not possible due to hardware limitations, that can be solved by doing the training over a powerful GPU.

4.5 Further Research

The last approach to simulate the ICC-Profiles is using *Convolutional Neural Networks* (CNN), where for each pixel is created a block $N \times N$ and it is sent to the CNN to calculate the output color. This model is heavier than the previous two, because there are a number of filters that takes different features of the input, which means that there will be more multiplications. Despite this approach might not be as useful as the previous two, it opens the possibility to a different kind of color transformations, which might be needed to take into account information of the Image.

The structure of the resulted CNN that was obtained in this project after some tests modifying the hyper-parameter is the following:

- Input block of size $11 \times 11 \times d$, where d is the number of channels of the input color space.
- *Convolutional* Layer using 10 filters with the size 3×3 doing a step of 2 pixels and using the *tanh* activation function.
- *Pooling* Layer using a filter 2×2 doing a step pixel in pixel
- A *Fully Connected Neural Network* of 3 hidden layers with 40, 11 and 5 pixels respectively using the *tanh* activation function.
- In the output layer there are d channels, depending on the output color space, using *Leacky ReLu* as activation function.

5 Quantitative Results

In this section it will be discussed all the results obtained from all the different tests and approaches found in terms of mean squared error, maximum error generated, and elapsed time to transform different images and using different devices.

5.1 Machine Learning Tools Comparison

In Table 1 there are the results obtained with the different models. To calculate the error of one color the Equ. 1 will be used but RGB colors will be used instead of Lab colors. To obtain the error generated by the model the MSE is calculated.

TABLE 1: Error of all the models tested

Model	**MSE** (ΔE)
Linear regression	28.4
RBF regression	11.5
Polynomial regression with 16 degrees	2.8
Neural Network from the papers	3.1

Despite the polynomial regression has the lowest error and the NN obtained a higher error, the degrees of the polynomial used to obtain such a small error are 16, which are a large number of degrees. A large number of degrees makes the model computationally heavy, whereas the NN can be improved by adding more neurons or layers or by changing the activation functions of the hidden and output layers.

5.2 The Neural Network architecture

After all the tests were done, we could see on the one hand that when using the non-square structures of NN this could not fit the data set and produced a high error. On the other hand, the square structure was giving good results if the number of layers was below 5. However, the models that generates the better results are the ones with 2 and 3 layers. For more details read appendix A.2.1.

In Fig. 2 we show the difference between using the tanh-ReLu activation function and using the Sigmoid-Lineal activation function. As explained before, the model using tanh-ReLu activation functions has a lower mean and a lower maximum. As a consequence in the following test these activation functions were used in the model.

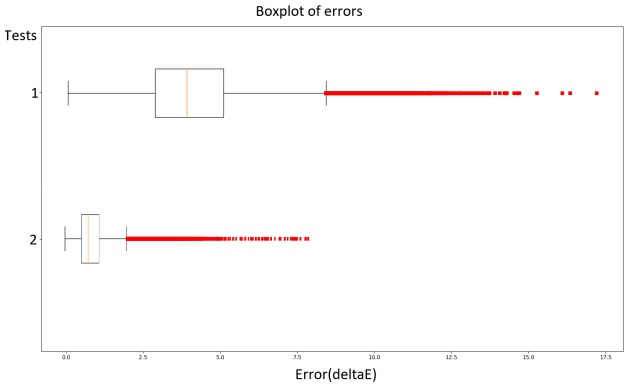


Fig. 2: Comparing the *Sigmoid-Linear*(1) model and the *tanh-Leaky ReLu*(2) model. The red line inside the box represents the mean error and the last right red point means the maximum error generated by the model.

5.3 Tuning the Data Set

In Fig. 3, we show how the colors go out of range by varying a given coordinate (R, G or B) and letting the others two fixed. All the colors generated were transformed with the LCMS and the NN. The results are plotted with the specific error below. In Fig. 3 we represent the obtained results with no modifications in the dataset. It is possible to see that the NN makes predictions over 255 as they are also made by LCMS.

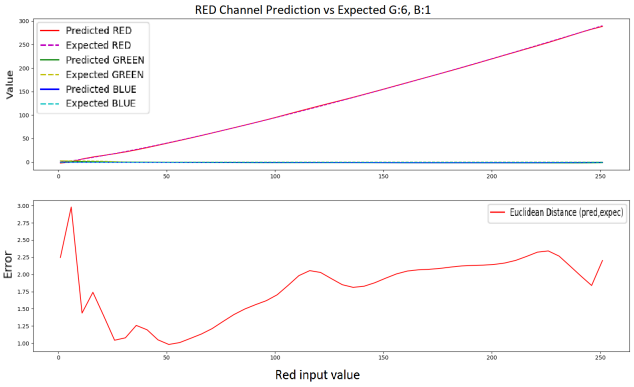


Fig. 3: Results with no tunning in the dataset. The output of the NN is equal to the output from the LCMS, although it goes over 255. In the first plot there are lines that are not visible, because they are one over the other, such as the red lines and the dotted line and also with the green and blue lines.

To solve the difficulty of having values over 255 we have implemented the simplest way first(1), so the NN will be trained with colors which are in the range of $[0...255]$. The result is plotted in Fig. 4.

The second implementation consists in removing the colors that are out of gamut while creating the data set that is used to train the NN.

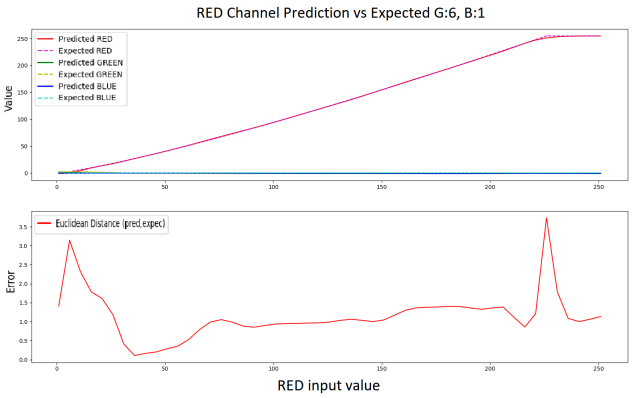


Fig. 4: Results with substituting values over 255 for 255. In the first plot it is represented the prediction of each color made by the network and the colors transformed with the LCMS. In the second plot the error of each color is represented.

Fig. 5 shows the plot with the results of the predicted colors with the NN and the expected colors calculated with the *LCMS*. It is possible to see that this model also fits the values below 255. Despite there is a large error with small color values, the green color line goes down instead of remaining straight. This is because there were colors deleted in the training set and now the network makes an inaccurate approximation based on what it saw in the training.

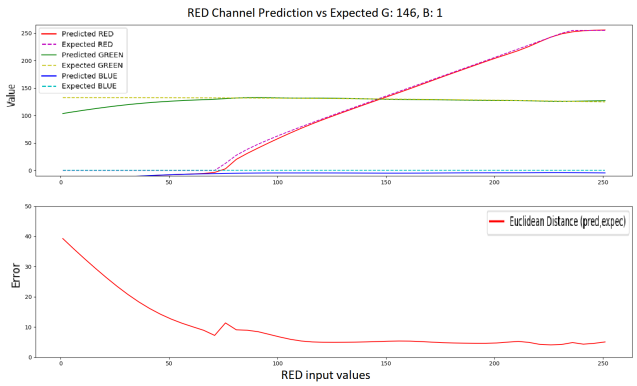


Fig. 5: Results with no colors out of gamut in the training. In the first plot it is represented the prediction of each color made by the network and the colors transformed with the LCMS. In the second plot the error of each color is represented.

5.4 Two Steps Color Management

All the models used to compute the CM in two steps were trained without reaching the overfitting as all the previous models trained. See appendix A.2.2 for more details.

Table 2 shows the results of the three models after being trained, where models 1 and 2 have a MSE close to zero and the maximum error is below 5. The error produced by these two models when transforming an image is low but perceivable as seen in section 6.

When using models 1 and 3, the error of model 3 is higher due to its non linearity. For more details check appendix A.2.3. As result, the image transformed using these two

models is higher than the previous combination and some colors do not have the same tonality as the expected image as can be seen in section 6.

TABLE 2: RESULTS OF THE 3 MODELS TO DO THE CM IN TWO STEPS.

Index	Transformation	MSE	Max error
1	aRGB to PCS	0.43	3.22
2	PCS to appRGB	0.76	4.6
3	PCS to devRGB	1.3	18.8

5.5 Comparing Created NN Models

Table 3 shows the errors that generate all the models tested. The one step model is the model that makes less error when doing the CM. As seen in Section 4.3, the best results are obtained when using the colors in the range $[0...255]$. The second model is the one that does the CM with the same steps that do the ICC-Profiles, in two steps. The first line corresponds to the transformation from *Adobe RGB* to *Apple RGB* and the second line corresponds to the transformation from *Adobe RGB* to *Device RGB*. When we combine two models the errors increase as seen in Table 3. Despite the quantitative results are not good enough, it is a first step of a future research on this field. Finally, the last model is the CNN. The errors obtained are much better than the Two Steps model but worse than the first model. However, this does not mean that this model is not useful, because more research is needed and it might be more accurate in other kind of transformations as explained in Section 4.5.

TABLE 3: COMPARING THE MEAN SQUARED ERROR AND THE MAXIMUM ERROR GENERATED FOR EACH MODEL.

Model	MSE	Max error	Data set
One step model	0.93	6.1	original
	0.91	5.3	$0 < 255$
	1.4	40.3	OOG
Two steps model	8.9	60.4	$0 < 255$
	30.5	120.4	$0 < 255$
CNN	2.5	11.5	$0 < 255$

These models are also compared in terms of time with the ICC-Profiles and not in terms of error due to the fact that the ICC-Profiles were used to create the data set used to train the different models. The devices used in this test are:

1. a CPU with 6 cores and 2 threads in each and 32GB of RAM.
2. the Quadro k2000, a GPU with 384 cores and 2GB of RAM.
3. the TITAN XP, a GPU with 3840 cores and 12GB of RAM.

In Table 4 we can see that the reference of the ICC-Profiles is in 0.15sec, this means that all the times below are an improvement. It can be seen that all the models, when transform the image in Fig. 6(a) using the CPU, the elapsed time is higher than using the ICC-profile. However, the advantage of using the NN is that they can be loaded into a GPU and make the calculations faster due to the parallelization done in the GPU. As expected, all the models are faster using any GPU, except the CNN that goes slower using the GPU's than using the CPU due that is not using properly the GPU's parallelization. This can happen, not all the NN works faster with a GPU vs a CPU.

The fastest model in CPU is the *Two Steps Model* due to the small size of the models even if the combination of two models has 4 hidden layers, but as they are calculated separately it goes faster. However, in GPU it goes 2 times slower than the *One Step Model* due to the parallelization goes faster for bigger models than two small models that calculates everything in two steps, which is caused by the I/O operations rather than the computational operations.

TABLE 4: COMPARING THE TIME ELAPSED OF EACH MODEL WHEN TRANSFORMING AN IMAGE FROM *Adobe RGB* TO *Apple RGB*

Model	size Image	Machine	Time (sec)
ICC-Profile	400×300	CPU	0.15
One step model	400×300	CPU	0.27
	400×300	Quadro	0.033
	400×300	TITAN xp	0.018
Two step model	400×300	CPU	0.14
	400×300	Quadro	0.073
	400×300	TITAN xp	0.038
CNN	400×300	CPU	0.53
	400×300	Quadro	4.88
	400×300	TITAN xp	2.48

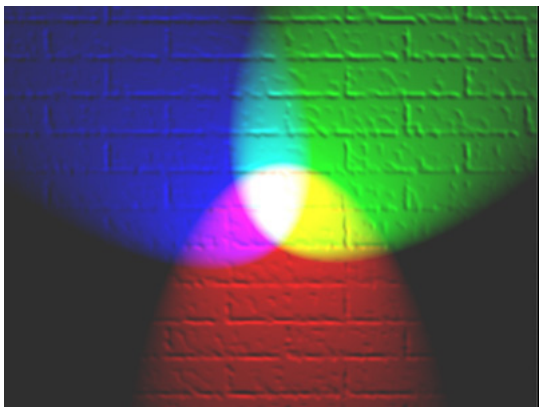
6 QUALITATIVE RESULTS

In this section we are going to use all the models created to transform the image in Fig. 6(a) in the Adobe RGB color space to the Apple RGB color space. All the images have been created with the *libtiff* [7] library in C++ and attach a profile and generate the "difference images" with *Adobe Photoshop* [3].

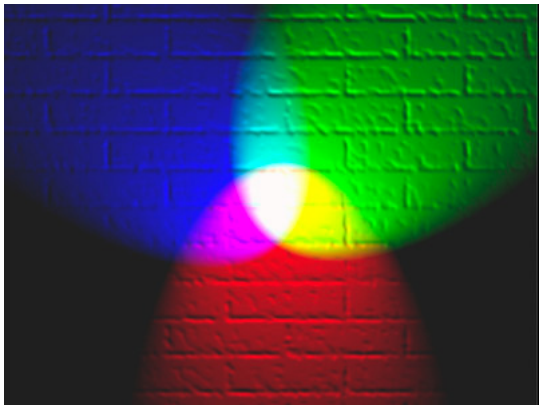
All the results are shown with the resulted image of the transformations and the difference image between the expected image and the resulted image. A gradient of colors has been applied to the difference image, as it can be seen at the right side of the image. The reddish the difference pixel is the higher the error is and the dark bluish the difference pixel is the lower the error is. The error of the difference images is 50 times the original, due to the small error obtained.

6.1 No Tuning Data Set

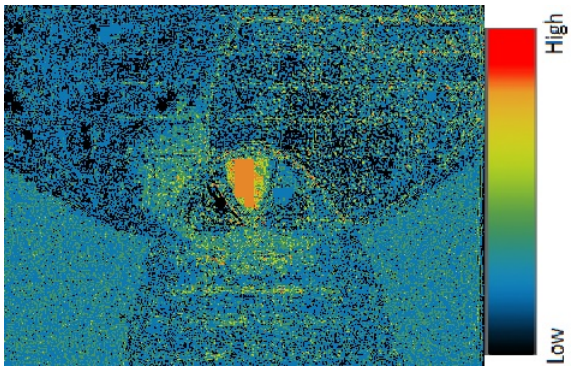
In Fig. 6 it is illustrated the transformed image with the model that was trained with the non-tunned data set. In the difference image is possible to see that there is a high error with the white colors, where the values of each channel are close to 255. There is a low error in the grey area, which is perceivable when comparing the resulted image with the expected image.



(a) Original Image



(b) Resulted Image



(c) Difference image

Fig. 6: (a) Original Image in *Adobe RGB* color space. (b) Resulting image without tunning in the data set. (c) Error multiplied by 50 and with a false color to highlight the error.

As the resulted image and original image are highly similar, in the following figures it is just shown the difference image that it will allow us to compare with the other models.

6.2 Limiting maximum value at 255

In Fig. 7 it is illustrated the transformed image with the model that was trained with data set that has the maximum value limited at 255. It is possible to see that the error in the with colors has decreased and it has also improved the grey area compared with the one in Fig. 6.

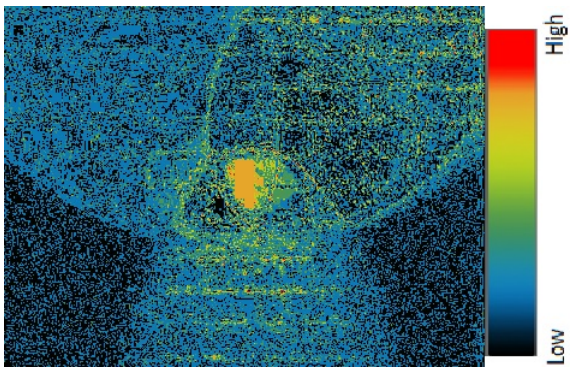
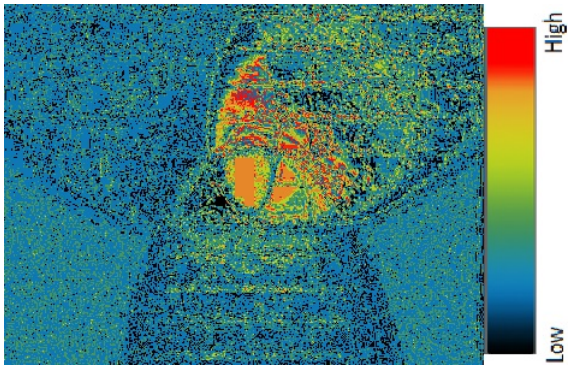


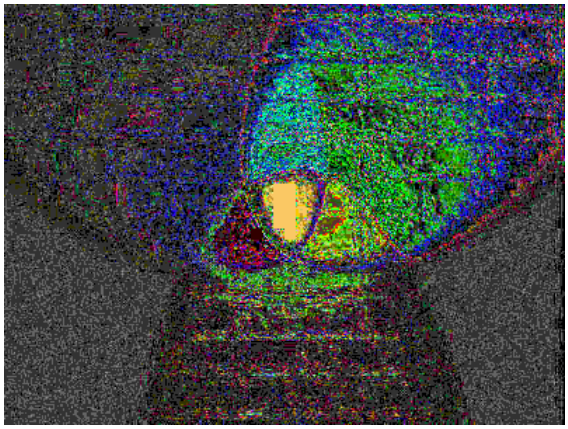
Fig. 7: Difference image with a false color of the resulted image using the data set limited to 255. The error was multiplied by 50 to highlight the error.

6.3 Removing Out Of Gamut Colors

In Fig. 8 it is illustrated the transformed image with the model that was trained with the data set with no out of gamut colors. In the difference image, it can be seen that there is a high error in the greenish area, and if we have a look to the image (b) it is possible to see that the channel that produce the higher error is the green and blue ones as it has been seen in Fig. 5



(a) Difference image



(b) Difference image in RGB

Fig. 8: Resulting error images using the *Out of Gammut* data set: (a) Error multiplied by 50 in a false color to highlight the error. (b) Error multiplied by 50 in RGB , where each pixel is the module of each channel. Useful to see which channels produce the maximum error in each pixel.

6.4 Two Steps Model

The qualitative results of the *Two Steps Model* can be seen in Fig 9. In the difference image it is possible to see that the area that has the highest error is the greenish area. This error is caused due to the color with more difference of gamut. The other colors have a low error in exception the grey colors which have medium-low error. The errors in the greenish area and in the grey area are perceivable when comparing the expected image with the resulted image.

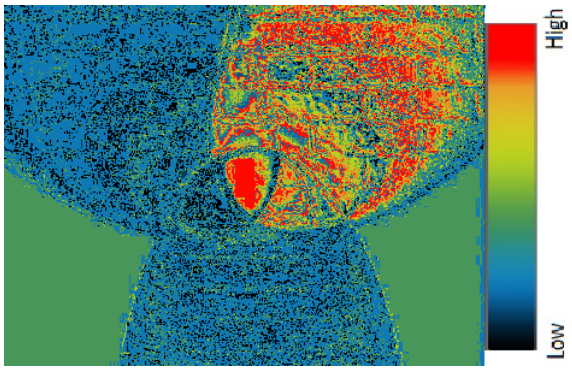


Fig. 9: Difference image with a false color of the resulted image using the *Two Steps Model*. The error was multiplied by 50 to highlight the error.

6.5 Convolutional Network

The last model tested is the *ConvNet* and the qualitative results can be seen in Fig 10. I can be seen in the difference image that there are no high errors in any of the image colors, which means that the convolution helped to fit better to the dataset. There is some error with the colors that have values close to 255 but are lower than the other models. However, the error in the grey area is higher than the model used to extract the results in Fig 7, and it is perceivable when compared with the expected image.

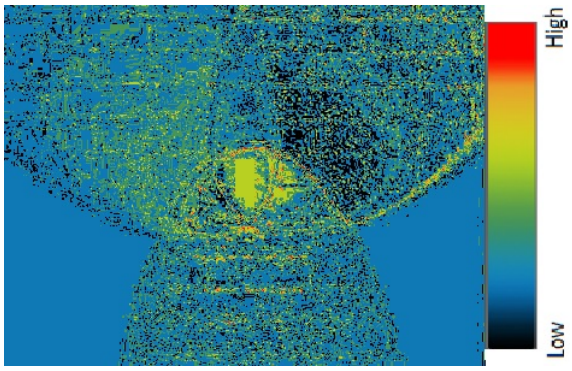
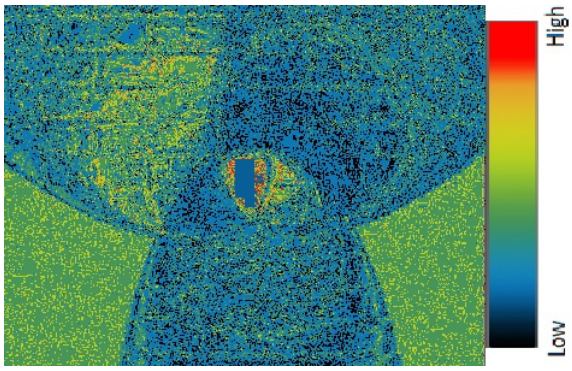


Fig. 10: Difference image with a false color of the resulted image using Convolutional Neural Network. The error multiplied by 50 to highlight the error.

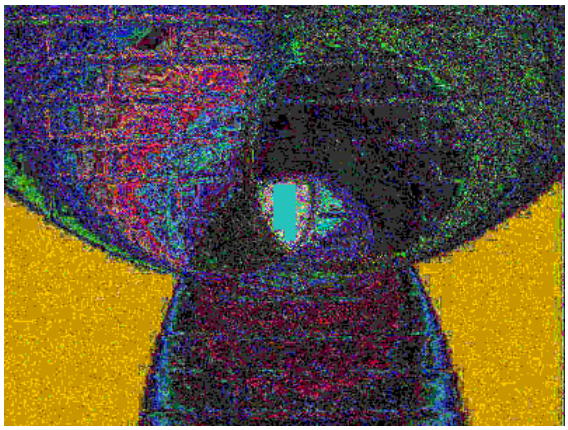
6.6 Trying Device Profile In One Step Model

Once known the architecture of the *One Step Model* NN to go from one RGB to another RGB color space, another model has been trained using the *Device RGB* profile mentioned in Section 4.4. In Fig. 11 it is shown the resulted image and the error produced between the resulted image and the expected image. The grey area has a medium-low error which is perceivable when comparing the resulted and the expected image. This high error is produced by the green and red channels, which explains that it appears a yellowish color in the difference image in RGB. In the blue area there is also a medium-low error produced by the channels blue and red.

The medium-low errors generated in this transformation, which are higher than the ones in Fig 7, are due to the *Printer Profile RGB* output using the LCMS is not as linear as is the *Apple RGB* and the NN has more difficulties to fit to the output expected.



(a) Difference image



(b) Difference image in RGB

Fig. 11: Resulting error images using the *Printer Profile RGB*: (a) Error multiplied by 50 with a false color to highlight the error. (b) Error multiplied by 50 in RGB , where each pixel is the module of each channel. Useful to see which channels produce the maximum error in each pixel.

6.7 General Observations

The model trained with the data set with no colors out of gamut generated the image with higher error as seen in Fig 8. It is possible to see in Fig 8 and in Fig 5 that the channel that generates a bigger error is the green one. The green color resulted to be the color that the Adobe RGB has a wider gamut area than the Apple RGB as it can be seen in Fig. 21. Whereas in Fig 6 and 7 the error is quite similar in all the image. In Fig 7 there has been an improvement in the white area and in the grey colors, which are not perceivable when comparing with the expected image.

In all the images seen in this section it was possible to see that the greyish and the whitish colors had a noticeable error. This is caused because the output from the profiles saturates at 0 and at 255 and is just before the saturation where

the NN fit worse the model and generates a high error as showed in Fig 12.

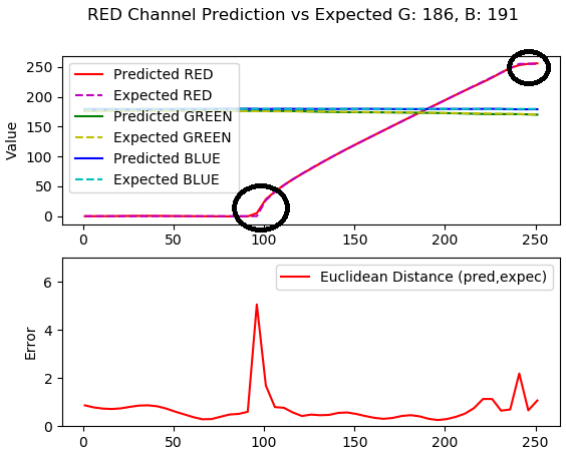


Fig. 12: Illustration to show why there are high errors in the greyish and whitish colors.

7 Conclusions

In this work we have seen that the CM can be carried out using Neural Networks and it has been proved that NNs give better results than other regression methods.

After knowing that it was possible to do the CM with NN, we searched the structure that better fits the data set. Resulted to be a Square-NN with 3 hidden layers and with 50 to 70 neurons per layer.

With the previous structure the results could have been improved, so we modified the activation functions of each layer of the NN. It resulted that by substituting the activation functions of the hidden layer from a *Sigmoid* to a *Tanh* function and the activation function from the output layer, from *Linear* to a *Leacky ReLu* function, the model was able to fit better and give better results.

To see how the model was predicting, we plotted the values predicted when modifying one dimension of the RGB input color and leaving the other two static. We realized that even the LCMS and the NN were giving values over 255 and to solve this the best option was to limit them to 255.

After making tests and transforming images to see the qualitative results, we found that using the first modification of the data set, the transformed image generated less error in the black colors, whereas the other colors remained similar.

We have seen that the higher errors are produced by the colors with values close to 255, the ones that are out of gamut and the greyish colors.

We have also seen that an architecture does not have to work for all the transformations between the same color space of colors due to there are some profiles that are more lineal than others.

To better approximate the steps used in the CM with profiles we used two NN models to transform the color values. We have seen that individually, each model generates a really low error. However, when we put them together this errors is added and is more noticeable in the greenish colors.

The final approach was to use a ConvNet, which resulted to generate low error when transforming images. However, it takes a lot of time to transform a small image. The ConvNet might not work for this kind of transformations but it might be useful in others such as adjusting the values of an specific area of the image that detects that it should have another values instead of the assigned ones.

Finally we can say that doing the transformation using NN is faster than doing it with the ICC Profiles due to it can be made over the GPU, which allows the parallelization of the transformation.

8 Lessons Learned

To start with the project I first had to understand how the transformations between color spaces were done [13], and to create the dataset that was used to train the NN. To create the dataset I used the LCMS library [10]. There is an API [11] and a tutorial [12] document to learn how to use the library, which is complex to understand and to use. Once I started to understand how everything works I saw the complexity that is behind the profiles with all the measures and the statistical process to define an ICC Profile.

To be able to create a NN I had to learn from scratch the Tensorflow library and the architecture used to build the diagrams. I read a lot of tutorials, from the Tensorflow API[2] and from other pages [9], to acquire knowledge necessary, from Tensorflow and Machine Learning in general, to be able to train the models used to solve the problem presented.

9 Acknowledgments

I want to thank the whole group of *Languages&Job management-2D* in HP Inc. for providing me all the material in order to accomplish the project and in particular to Jaume Vergés, who explained me how the Color Management works and who gave me some feedback of the results obtained along the project. Finally, I would also like to thank Fernando Vilariño for giving me advice on how to improve the reports and for the opportunity to talk with a color specialist and to compute some tests over the NVIDIA TITAN GPU.

References

[1] Scikit learn library. `http://scikit-learn.org/stable/index.html`. Last access: 06-03-2018.

[2] Tensorflow library. `https://www.tensorflow.org/`. Last access: 14-06-2018.

[3] Adobe. Adobe photoshop cc 2018. `https://www.adobe.com/es/products/photoshop.html`. Last access: 15-06-2018.

[4] Yan Li Baohui Xu and Jie Zhang. Research and comparison of display color conversions based on standard of icc color management. *2009 9th International Conference on Electronic Measurement and Instruments*, 2009.

[5] Zhi Chuan and Zhou Shi-Sheng. Research on color space transformation model between rgb and l*a*b*

based on bp neural network. *International Conference on Computer Science and Software Engineering*, 2008.

[6] Cao Congjun and Sun jing. Study on color space conversion between cmyk and cie l*a*b* based on generalized regression neural network. *International Conference on Computer Science and Software Engineering*, 2008.

[7] LibTiff group. Libtiff - tiff library and utilities. `http://www.simplesystems.org/libtiff/`. Last access: 27-04-2018.

[8] Matplotlib. Matplotlib api. `https://matplotlib.org/api/index.html`. Last access: 09-06-2018.

[9] Andrew Ng. Machine learning by andrew ng. `https://www.coursera.org/learn/machine-learning`. Last access: 14-06-2018.

[10] Martí Maria Saguer. Little cms library. `http://www.littlecms.com/`. Last access: 26-03-2018.

[11] Martí Maria Saguer. *Little CMS Engine API 2.9*. 2017.

[12] Martí Maria Saguer. *Little CMS How to use the engine in your applications*. 2017.

[13] José E. Pereira Uzal. *Gestión del color en proyectos de digitalización*. Marcombo, 1st edition, 2013.

APPENDIX

A.1 Support figures

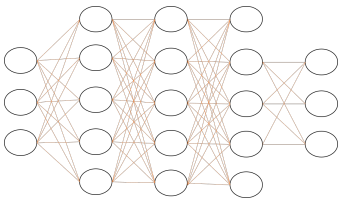


Fig. 13: Example of a square NN

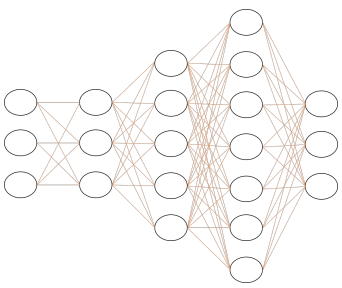


Fig. 14: Example of an increase NN

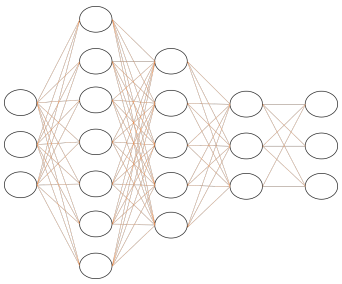


Fig. 15: Example of a decrease NN

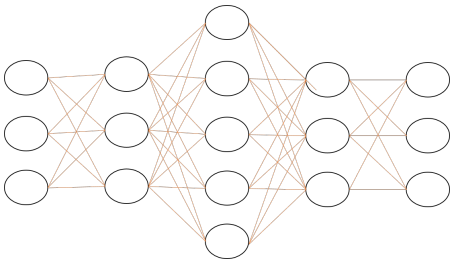


Fig. 16: Example of an Inc-Dec NN

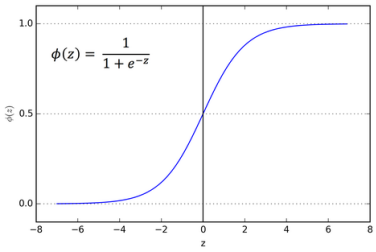


Fig. 17: Representation of a sigmoid function.

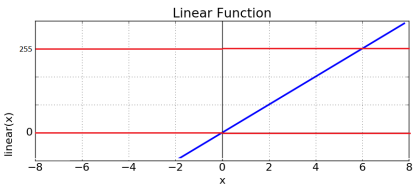


Fig. 18: Representation of the linear function and the color range.

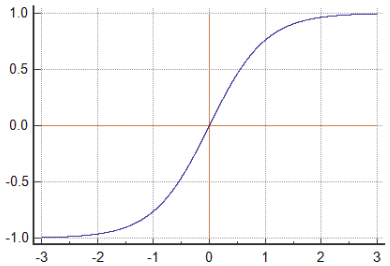


Fig. 19: Representation of the tanh function.

A.2 Representation of results

The following plots have been created using *The Matplotlib* library [8].

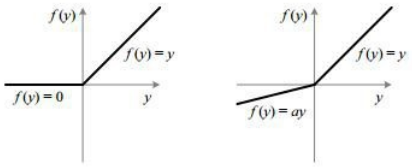


Fig. 20: Representation of the ReLu and Leaky ReLu functions respectively.

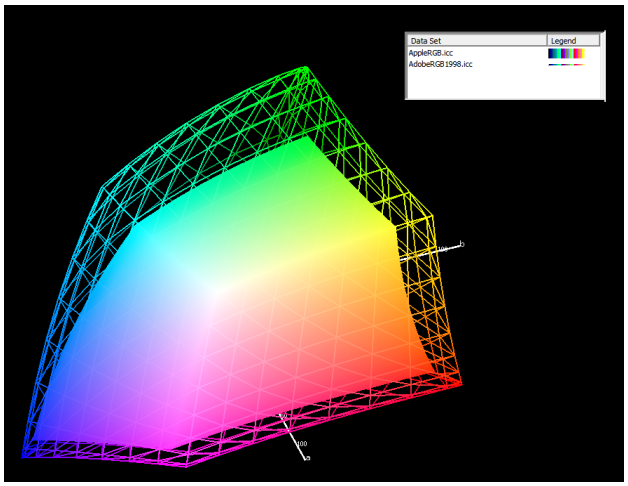


Fig. 21: Adobe RGB and Apple RGB gamut comparison over the LAB color space

A.2.1 Modifying Neurons and Layers

The first test used the Square NN model and in Fig. 22 it can be seen that all the NN with 5 and 6 neurons give really poor results and those with less than 5 layers give similar results. However, the more layers and neurons the slower the NN is. In Fig. 22 it can be seen that the NN with 3 layers gives the best results.

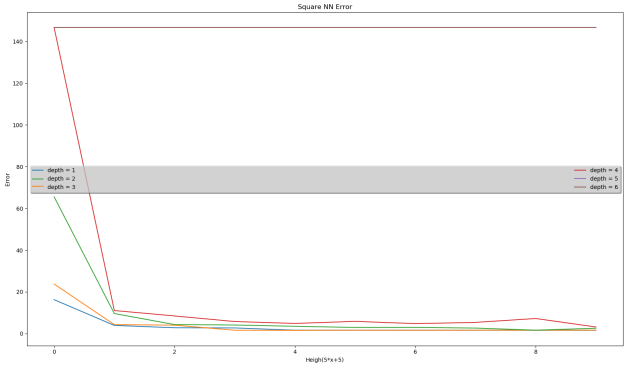


Fig. 22: Square Network results by modifying the width and the depth

The following tests used the Increase, Decrease and Increase-Decrease Networks, respectively the three results were all similar. This is why only one plot is shown in Fig. 23. This plot is the results obtained for each Network. The tests of this Networks were configured to be 100K epochs, where one epoch is one iteration of training, to see if the results would change. Instead of improving, the results became even worse, which means that this three structures are not suited for the purpose of CM.

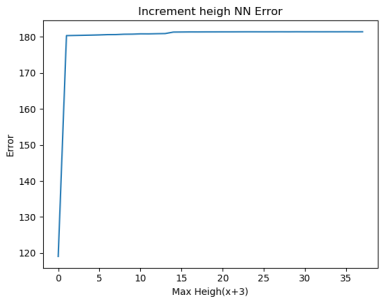


Fig. 23: Increment Network results by modifying the maximum number of neurons.

A.2.2 Two Steps Model Training

In Fig. 24, 25 and 26 it is possible to see that the models were trained until the mean squared error was close to zero, and without doing overfitting with the training data set.

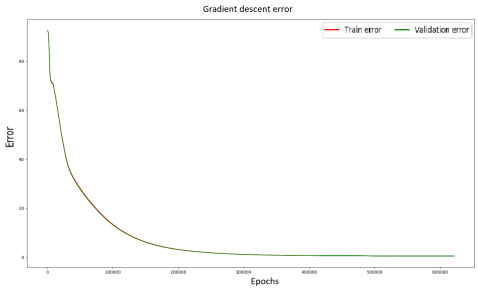


Fig. 24: Gradient descent of the model that goes from *Adobe RGB* to the PCS, where it is possible to see how the error in the training and in the validation data set decreases along the iterations.

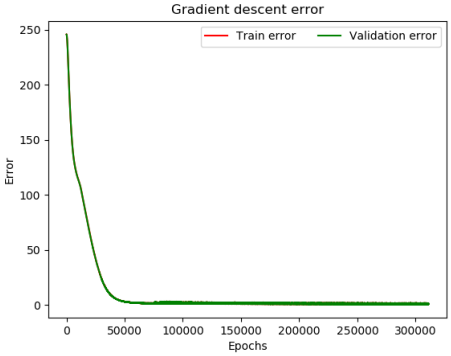


Fig. 25: Gradient descent of the model that goes from the PCS to appRGB, where it is possible to see how the error in the training and in the validation data set decreases along the iterations.

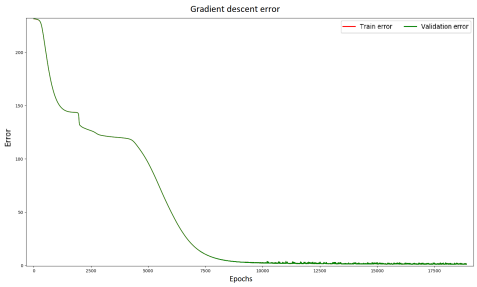


Fig. 26: Gradient descent of the model that goes from the *Adobe RGB* to *Apple RGB*, where it is possible to see how the error in the training and in the validation data set decreases along the iterations.

A.2.3 Non Linearity Printer Profile RGB

In Fig 27 it is possible to see that there are no channels that follow a straight progression. As result of this non-linearity the NN does not fit the data set without generating high errors.

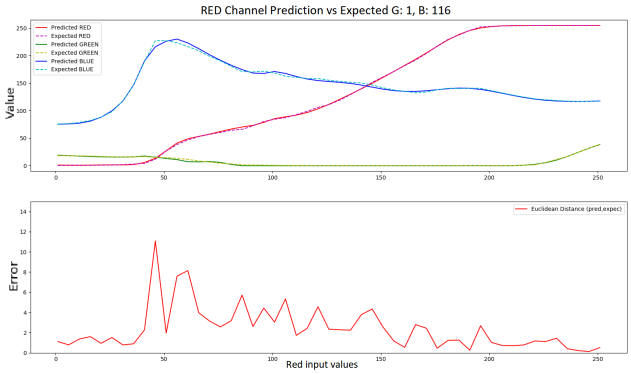


Fig. 27: Output of the device RGB values and how the NN fits poorly to the non linearity of the values.