Color Characterization of Displays using Neural Networks

Ujjayanta Bhaumik[1,*,†], Rik Marco Spieringhs[2,†] and Kevin A. G. Smet[3,†]

[1,2,3] *Wavecore/Light&Lighting Laboratory, KU Leuven, Belgium*

Abstract

When studying human color perception using displays, accurate and reproducible presentation of color stimuli is paramount, which requires color characterization of the display. Often simple (non-linear) models, such as the Gain-Offset-Gamma (GOG) model, or low-resolution look-up tables (LUT), both based on a limited number of optical measurements, work well enough. However, some displays show a much more complex relationship between RGB input and color output, requiring more complex models or high-resolution LUTs based on a large number of measurements. In this paper, as the first step in a larger study, the feasibility and performance of neural networks (NN) for color characterization of a simulation display following a simple GOG model are explored and compared to a LUT-based method. Statistical analysis showed that the NN-method performed significantly better than the LUT model in terms of predicting the required RGB device input values that generate a set of target (device output) XYZ tristimulus values. In fact, to achieve the same accuracy, the number of training points can be substantially reduced for the NN-method compared to the LUT-method. Furthermore, the neural network is also more than 20 times faster than the look-up table in performing display characterization.

Keywords

Display characterization, Neural Network, Look-up table

1. Introduction

Color characterization is the process of converting a device-dependent color space to a device-independent space like the XYZ or LMS. The necessity of characterization arises from the primaries that a device uses to produce colors. As the primaries are different in each device, these intrinsic differences give rise to device-dependent color spaces [1]. In different cathode ray tube (CRT) monitors, for instance, the red, green, and blue phosphors are physically different resulting in minute differences in color produced by them for the same input value. So, if a person is sending a particular input RGB color, for instance, $(200, 0, 200)$ to two different devices, depending on how the primaries are in both devices, the output on the display can be different. A device-independent color space, on the other hand, would always produce the same color output for a particular input. For a color characterized device, one would know the relation between the device-dependent color space and device-independent color space.

The 11th Colour and Visual Computing Symposium, September 08–09, 2022, Gjovik, Norway

*Corresponding author.

†These authors contributed equally.

ujjayanta.bhaumik@kuleuven.be (U. Bhaumik); rik.spieringhs@kuleuven.be (R. M. Spieringhs); kevin.smet@kuleuven.be (K. A. G. Smet)

0000-0003-1498-7726 (U. Bhaumik); 0000-0003-4956-837X (R. M. Spieringhs); 0000-0003-3825-6274 (K. A. G. Smet)

The problem of characterization can be treated as finding the relation between two sets of points in $\mathbb{R}^3$. A function is to be determined from XYZ color space to RGB color space so that one can determine which RGB values should be sent to the headset to display the correct XYZ output. For instance, to achieve a particular output $(X_1 = 19.2, Y_1 = 21, Z_1 = 73.2)$, the fitted function would predict $(R_1, G_1, B_1) = f(19.2, 21, 73.2)$. This triplet (R_1, G_1, B_1) can then be sent to the device to obtain the output as (X_1, Y_1, Z_1).

This function can be determined in a variety of ways, for instance, using a Gain-Offset-Gamma (GOG) model or look-up tables [2]. This paper presents a novel approach based on simulated datasets of different sizes to thoroughly compare methods that use multilayer perceptron based neural networks with a varying number of hidden layers to determine the color characterization function and traditional look-up tables. Multilayer perceptrons have been used for solving problems like pattern recognition and interpolation and were an improvement over the simpler predecessor perceptron suitable for linearly separable problems [3]. They provide the necessary complexity for predicting more difficult non-linear functions. The next sections describe the characterization techniques relevant to this work in more detail: GOG model, look-up-table, and neural networks.

1.1. Gain-Offset-Gamma model

The Gain-Offset-Gamma (GOG) model is one of the several ways of characterizing displays. For an overview of other methods, one can refer to Brainard et. al. [4]. It consists of 2 stages: the first step is a non-linear transformation to convert digital RGB values to linear RGB values, and the second step converts the linear RGB values to tristimulus XYZ values using a matrix multiplication [5].

The first stage, which is also a forward transformation from digital to linearized RGB values, is represented by a simple gamma transform [6]. Let $(r, g, b) \in [0, 1]$ be the digital pixel values to be sent to the display. Then the linearized values are given by:

$$R = r^\gamma \tag{1}$$

$$G = g^\gamma \tag{2}$$

$$B = b^\gamma \tag{3}$$

Here, $0 \leq r, g, b \leq 1$

After linearization, it is now possible to represent the device-independent (X,Y,Z) tristimulus values as a linear combination of the device-dependent linearized (R,G,B) values using a 3 x 3 matrix multiplication:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = T \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4}$$

where T is the transformation matrix from linear RGB to tristimulus XYZ values. If one measures the tristimulus values for maximum red, maximum green, and maximum blue, the equation for T is given by equation (5).

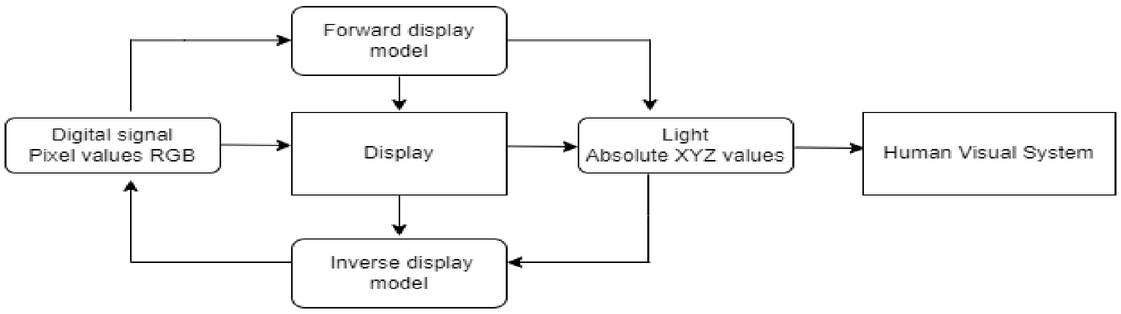

Figure 1: A full flow chart of general display characterization depicting the forward and backward display models. The forward model guides the conversion of the input digital signal values to output light in terms of tristimulus XYZ values while the backward model does the reverse [6].

$$T = \begin{bmatrix} X_{r,maximum} & X_{g,maximum} & X_{b,maximum} \\ Y_{r,maximum} & Y_{g,maximum} & Y_{b,maximum} \\ Z_{r,maximum} & Z_{g,maximum} & Z_{b,maximum} \end{bmatrix} \tag{5}$$

Although it is typically more accurate to determine the 9 matrix coefficients based on a least-squares minimization algorithm using a sufficiently large number of measured (RGB, XYZ) pairs.

If we also consider the minimal ambient light, the full equation becomes:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} - \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{ambient} = T \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{6}$$

It represents the fact that even for R=G=B=0 (no display output), still some non-negative XYZ values might be measured due to the ambient light. This is also sometimes referred to as display flare.

The whole characterization process is explained in Figure 1.

1.2. Look-up table (LUT)

A look-up table (LUT) is a dictionary that allows one to search for a value corresponding to a key. Look-up tables have been used historically since the time of Babylonian mathematics when they used them to calculate logarithmic tables [7]. In the context of display color characterization, a look-up table would store (RGB, XYZ) pairs which can be used to find the XYZ value for a characterized display corresponding to a RGB input or vice versa. An instance is shown in Figure 2.

When converting from device (r, g, b) to (X, Y, Z) using a fully populated LUT (i.e. one which has a pair for every possible r,g,b combination), the conversion process is reduced to a simple look-up. However, given that there would be $16777216 (= 256^3)$ entries loading the LUT into memory, and especially the look-up process might be quite time consuming. The

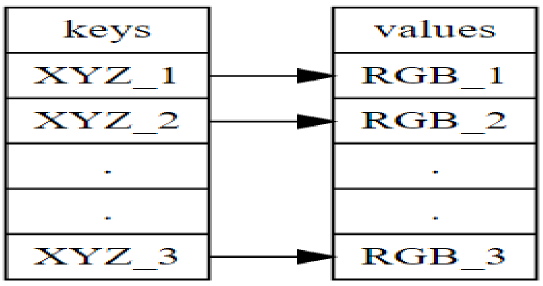

Figure 2: A look-up Table is a dictionary of keys and corresponding values. This example shows a look-up table that has XYZ tristimulus values and corresponding RGB values for a device. In this case, for getting a particular RGB output, one would find the corresponding XYZ value from the table and use that as input to the device.

actual measurement time required to generate the LUT would also be huge. Look-up tables are therefore usually determined by measuring pairs for only certain (r, g, b) combinations, whereby intermediate values are interpolated. Interpolation would almost certainly also be required for the inverse conversion, i.e. when going from device-independent (X, Y, Z) to device-dependent (r, g, b), as its highly unlikely that the desired (X, Y, Z) is one of the values in the LUT!

1.3. Neural Networks

This paper presents a neural networks based method to address the problem of accurate color characterization for displays. Cheung et. al. used neural networks and polynomial functions for camera characterization and both methods were found capable of producing similar results [5]. Vrhel et. al. used neural networks to calibrate color scanners and found that they performed better over polynomial-based methods [8]. Usui et. al. used neural networks to do color transformations for color management systems. This was done to take care of colors produced by different media and Usui et. al. showed that a neural network of 3 layers can be used as a powerful tool for such purpose [9].

Neural networks were used by Climent et. al. to test the accuracy of LCD displays [10] and they notified the appropriateness of neural networks in learning XYZ-RGB relations. Different groups of techniques are used for colorimetric characterization like methods that try to model the color physically assuming independence between channels [11],[12],[13].

Statistical techniques like Linear Color Correction (LCC) [14], polynomial regression [14] and geometric methods like 3D thin-plate spline [15] are some other methods used for colorimetric characterization of displays but methods like LCC can map linearized RGB to XYZ with high errors [14]. This paper used a feedforward network to perform display color characterization. In a feedforward neural network, the propagation of information is only in the forward direction and there are no cycles or loops. An instance of a feedforward network is shown in Figure 3.

A multilayer feedforward network can approximate any continuous function with just one hidden layer. This is referred to as the universal approximation theorem[8]. The mapping of XYZ color space to RGB color space or vice versa can be achieved with such a network. In this

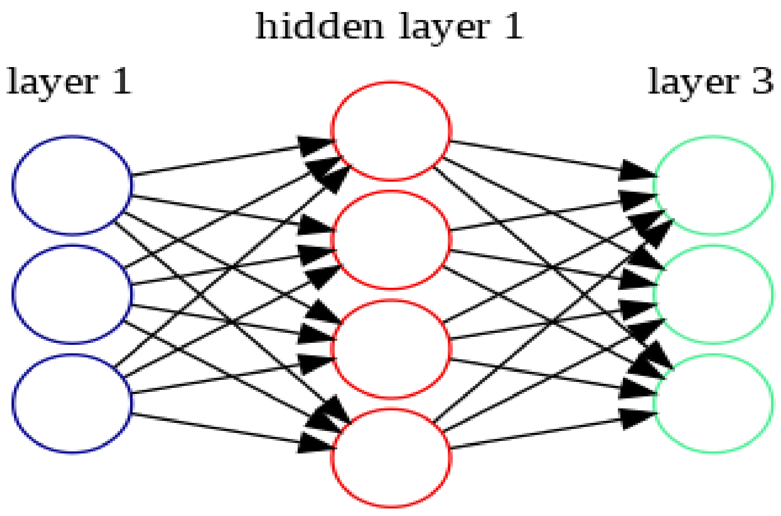

Figure 3: A simple feedforward neural network with 3 layers. The input layer has 3 nodes, the hidden layer 4, and the output 3. The arrows show the propagation of data and it's always one way propagation.

paper, we present a GOG model, a look-up table, and our neural network approach to perform characterization.

2. Setup

For the purpose of this research, an Alienware m15 laptop with 16GB RAM, 64 bit Windows operating system, and Intel(R) Core(TM) i7-8750H CPU at 2.20 GHz was used. The simulation was done in Python using the Luxpy library [16].

3. Methodology

3.1. Data Simulation

A virtual display was simulated using a Gain Offset Gamma model to generate $RGB \equiv XYZ$ pairs. It has been assumed that the display is perfectly characterized by the gain offset gamma model (no measurement errors). Several data sets, composed of $(r, g, b) - (X, Y, Z)$ pairs, were simulated to train the neural network, generate the look-up table, and test both methods. Both methods were always trained (for the LUT-based method this means generation of the LUT) and tested using the same data sets to ensure a fair comparison of their performance.

3.1.1. Sampling of RGB space

The first step of data generation for the training set included the sampling of RGB space into equal voxels while ensuring that the axes of the RGB color space, and particularly the maximum

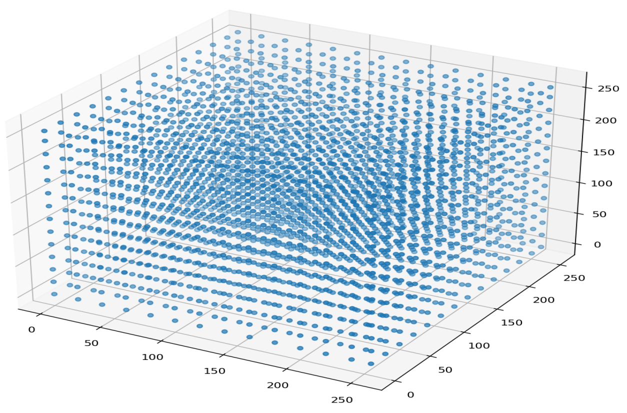

Figure 4: RGB samples generated using regular grid used for data simulation (n = 20).

value (255), were also sampled. The sampled (r, g, b) data points are represented by the set S^3:

$$S = \{x : x \in \mathbb{R} \;\&\; n|x\} \cup \{255\}, 6 \leq n \leq 20 \tag{7}$$

The red, green and blue axes are sampled uniformly due to the fact that equal sampling is the baseline or the simplest approach and it helps in representing non-additivity better. If any of the red, green or, blue channels contribute to the non-additivity, then the equal sampling makes sure an equal impact of all the channels is present. An example of the sampling with $n = 20$ is shown in Figure 4. Here, n is called the LUT increment, and the data set composed of the sampled RGB space is henceforth referred to as the RGB cube.

RGB cubes for neural network training and LUT generation were created for LUT increments of: $[6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]$ which corresponded to [79271, 50676, 32890, 24398, 17699, 13927, 10736, 8105, 6916, 5008, 4175, 3455, 3402, 2801, 2268] number of data points in the cube.

The (r,g,b) values for the test set were generated using Numpy's random generator function. The same test set was used in combination with all training sets and was composed of 10000 points. To complete the training and test sets and generate (r,g,b)-(X,Y,Z) pairs, their (r,g,b) values were converted into (X,Y,Z) tristimulus values using the GOG model.

3.2. Neural Network Training

The $RGB \equiv XYZ$ training data generated using the simulated algorithm are fed to a multilayer perceptron (coded with the Perceptron function from scikit-learn sklearn.linear_model class [17]) with the following parameters:

- hidden layer = [number of hidden layers]
- relu activation function ($f(x) = maximum(0, x)$)
- maximum iteration = 100000
- adaptive learning rate: adam optimizer

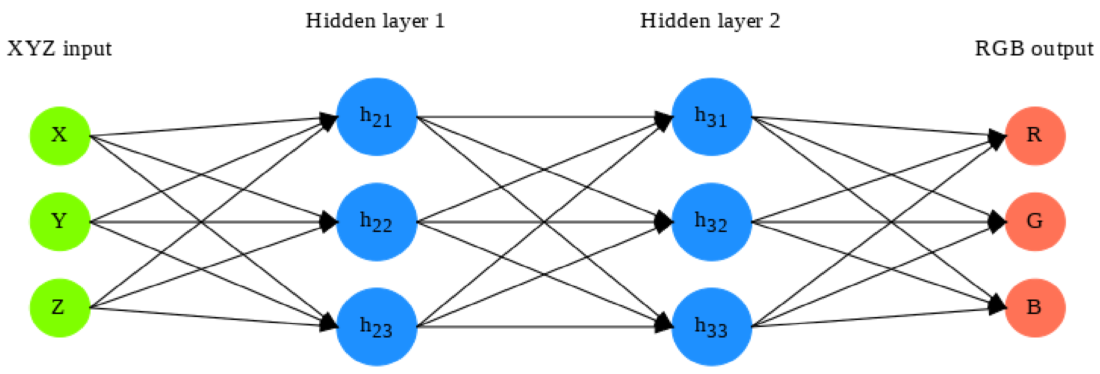

Figure 5: An instance of a neural network with 2 hidden layers. The input is (X, Y, Z) and the output is linearized (R, G, B).

The number of hidden layers is varied from the set [10, 20, 40, 80, 130, 160, 400, 460, 600] to test the performance of the neural network. The Adam optimizer uses an adaptive learning rate for optimizing stochastic objective functions [18].

3.3. Look-up table

The points in RGB cubes for each LUT increments of: [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20] were converted into XYZ points using the simulation algorithm. This resulted in (r, g, b), (X, Y, Z) pairs for each LUT increment.

For the LUT, predicting the value of input involved a few steps. First for finding the value of the new element, one needs to find the nearest neighbors and assign a value based on the values of the nearest neighbor from the table. This requires querying the LUT for the nearest neighbors. The number of neighbors considered for different experiments ranged in the set [1,2,3,4,5,6,8,10,12,14,16,18,20,25]. The second step involved using the cKDtree function from scipy for querying the different LUTs [19]. Then, the distances of the input point from all the nearest neighbors returned by the algorithm are calculated.

The predicted value for the input is calculated using a weighted average of the values of the nearest neighbors. The weights are calculated using the inverse squared distances of the query input point from the nearest neighbors. So, if a key is closer in distance to the input query point, it contributes more to determining the output value.

The predictions using the same test set are carried out using both the neural network and LUT and the results are presented in the next section.

4. Results & Discussion

A GOG model was used as the perfect model to generate the ground truth (RGB, XYZ) pairs. The forward model to go from display (r, g, b) to linear (R, G, B) is a simple Gamma function.

The transformation matrix given in Eq. (5) and (6) allows conversion from RGB to XYZ and

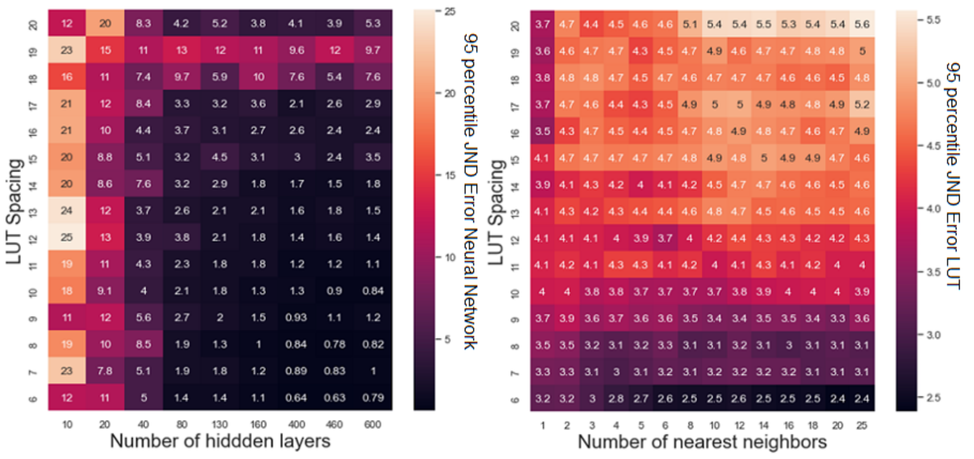

Figure 6: A side-by-side plot of Just noticeable difference (JND) errors for both Neural Network and LUT-based method. 1 JND corresponds to an Euclidean distance of 0.0033 $u'v'$ units in the CIE 1976 $u'v'$ color space [20]. For the figure on the left, the x-axis represents the number of hidden layers of the neural network, the y-axis the LUT increment. For the figure on the right, the x-axis represents the number of nearest neighbors for LUT, the y-axis the LUT increment. The numbers inside the individual boxes are the 95 percentile JND error produced by the LUT-based method. The color bar on the right shows the magnitude of error in each graph.

vice versa. The forward transformation matrix (M) to convert linear RGB to XYZ and the reverse transformation matrix (N) to convert XYZ back to linear RGB are calculated and given below.

$$M = \begin{bmatrix} 1.0858e+02 & 4.6817e+01 & 3.6931e+01 \\ 5.1134e+01 & 1.3333e+02 & 1.8603e+01 \\ 1.6299e+00 & 1.5691e+01 & 2.0352e+02 \end{bmatrix} \tag{8}$$

$$N = \begin{bmatrix} 1.0954e-02 & -3.6517e-03 & -1.6539e-03 \\ -4.2343e-03 & 8.9932e-03 & -5.3700e-05 \\ 2.3872e-04 & -6.6410e-04 & 4.9309e-03 \end{bmatrix} \tag{9}$$

Error maps are generated both for the Neural Network method and the LUT-based method to compare their performances. For both methods, the LUT increments were kept consistent. Errors were calculated for the Neural Network method at various numbers of hidden layers and the LUT-based method at various numbers of nearest neighbors. The error maps are shown in Figure 6. The neural network performs extremely well with LUT increments of less than 16 and at a number of hidden layers more than 100. Even at finer resolution with a lower LUT increment, the performance of the LUT-based method is worse compared to the Neural Network based method.

Considering the error graph in Figure 6., there are recommendations for the number of hidden layers of the neural network where it performs substantially better than the LUT-based method.

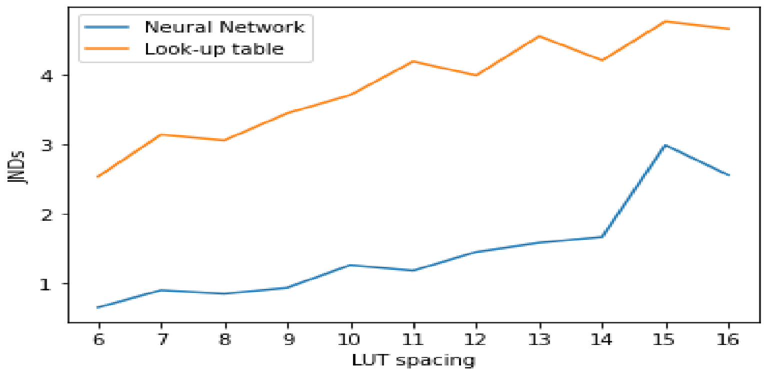

Figure 7: A comparison of the 95 percentile errors produced by the neural network (with 460 hidden layers) and look-up table (with nearest neighbors equal to 10). The neural network can be clearly seen to produce lower Just Noticeable Difference (JND) errors in CIE 1976 $u'v'$ space compared to the look-up table at different LUT increments.

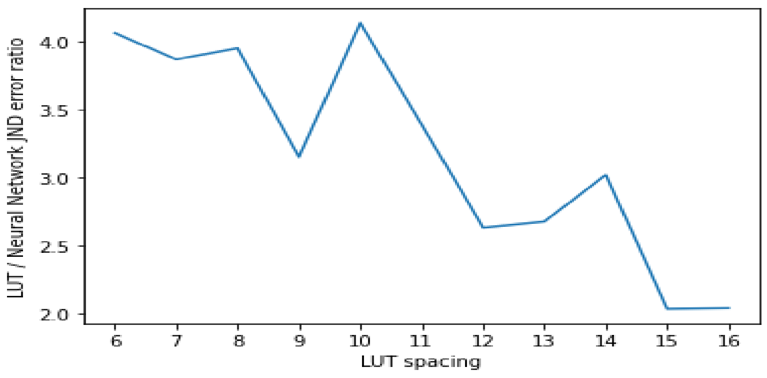

Figure 8: Relative performance of the neural network (with 460 hidden layers) and look-up table (with nearest neighbors equal to 10).

Figure 7. shows the performance of the neural network at the number of layers equal to 460 and a look-up table with the number of nearest neighbors equal to 10.

It can be clearly seen that the neural network performs substantially better in terms of reduced JND errors in the CIE 1976 $u'v'$ color space. Figure 8. shows the relative performance of the neural network and the look-up table.

In the worst-case scenario at LUT increment equal to 16, the neural network is still about 2 times better than the look-up table. On the other hand, in the best-case scenario, when LUT increment is 10, the neural network performs about 4 times better.

The neural network also provides a considerable advantage over the lookup table in terms of the time required to predict an input. The neural network performs substantially faster ($\times 22$) with an average search time of 0.001 seconds compared to 0.022 seconds for the look-up table.

5. Conclusion and Future Work

In this paper, a comparison of the performance of display color characterization methods based on a neural network and a look-up table was carried out. The analysis showed that on average the neural network was substantially better than the look-up table. Prediction time was also better for the neural network which provided over 22 times speed-up compared to the look-up table method.

In the future, we propose to test the neural network and look-up-table methods on real measured displays, some of which that have more complex relationships between the RGB input values and the measured XYZ tristimulus output, due to, for example, additivity failure. Using a look-up table for high resolution images becomes computationally expensive and a neural network might significantly help in obtaining better real-time predictions, while also offering the possibility of reducing the number of characterization measurements for the same accuracy.

References

[1] M. D. Fairchild, Color appearance models, John Wiley & Sons, 2013.

[2] R. S. Berns, Methods for characterizing crt displays, Displays 16 (1996) 173–182.

[3] L. Noriega, Multilayer perceptron tutorial, School of Computing. Staffordshire University (2005).

[4] D. H. Brainard, D. G. Pelli, T. Robson, Display characterization, Signal Process 80 (2002) 2–067.

[5] V. Cheung, S. Westland, D. Connah, C. Ripamonti, A comparative study of the characterisation of colour cameras by means of neural networks and polynomial transforms, Coloration technology 120 (2004) 19–25.

[6] R. R. Hainich, O. Bimber, Displays: fundamentals & applications, AK Peters/CRC Press, 2016.

[7] M. Campbell-Kelly, M. Croarken, R. Flood, E. Robson, et al., The history of mathematical tables: from Sumer to spreadsheets, Oxford University Press, 2003.

[8] M. J. Vrhel, H. J. Trussell, Color scanner calibration via a neural network, in: 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258), volume 6, IEEE, 1999, pp. 3465–3468.

[9] S. Usui, Y. Arai, S. Nakauchi, Neural networks for device-independent digital color imaging, Information Sciences 123 (2000) 115–125.

[10] J. Prats-Climent, L. Gòmez-Robledo, R. Huertas, S. García-Nieto, M. J. Rodríguez-Álvarez, S. Morillas, A study of neural network-based lcd display characterization, in: London Imaging Meeting, volume 2021, Society for Imaging Science and Technology, 2021, pp. 97–100.

[11] P. Colantoni, J.-B. Thomas, J. Y. Hardeberg, High-end colorimetric display characterization using an adaptive training set, Journal of the Society for Information Display 19 (2011) 520–530.

[12] W. B. Cowan, N. Rowell, On the gun independence and phosphor constancy of colour video monitors., COLOR reSearch and application 11 (1986) s34–s38.

[13] D. H. Brainard, Calibration of a computer controlled color monitor, Color Research & Application 14 (1989) 23–34.

[14] G. D. Finlayson, M. Mackiewicz, A. Hurlbert, Color correction using root-polynomial regression, IEEE Transactions on Image Processing 24 (2015) 1460–1470.

[15] P. Menesatti, C. Angelini, F. Pallottino, F. Antonucci, J. Aguzzi, C. Costa, Rgb color calibration for quantitative image analysis: The "3d thin-plate spline" warping approach, Sensors 12 (2012) 7063–7079.

[16] K. A. Smet, Tutorial: The luxpy python toolbox for lighting and color science, Leukos 16 (2020) 179–201.

[17] Sklearn.linear_model.perceptron, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html, 2022. Accessed: 2022-10-13.

[18] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[19] scipy.spatial.ckdtree, https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html, 2022. Accessed: 2022-10-13.

[20] J. Zhang, Y. Meuret, X. Wang, K. A. Smet, Improved and robust spectral reflectance estimation, Leukos 17 (2021) 359–379.