UDC 004.92 **DOI: https://doi.org/10.30837/PMW.2025.T2.005**

A STUDY ON THE USE OF ARTIFICIAL INTELLIGENCE FOR BUILDING DIGITAL PROFILES OF PRINTING EQUIPMENT

Azarenkov V.I.
Ph.D., Associate Professor, Department of System Analysis
and information and analytical technologies,
National Technical University
"Kharkiv Polytechnic Institute"
Deineko Zh.V.
PhD, professor, department of Media Systems and Technologies
Kharkiv National University of Radio Electronics

Abstract. *The issue of researching the possibilities and identifying effectiveness of the approximation application of measured data using artificial neural networks in constructing color profiles of printing equipment is raised. Problems and ways to solve them are highlighted. An analysis of the features of converting one color model into another based on the construction of ICC profiles was conducted.*

Key words: *COLOR MANAGEMENT SYSTEM, COLOR PROFILE, NEURAL NETWORKS, CLUT TABLES, INPUT, OUTPUT AND DISPLAY DEVICES.*

Introduction

Historically, color management has been a laborious and expensive process in the printing and prepress industries. There have been intense discussions over the years about how to address this issue.

In the traditional approach to a color management system, the characterization of the entire equipment chain, consisting of a scanner, image processing program, monitor, and output device, is possible only if the composition of such a chain is unchanged. Only when the composition of the chain is finally determined, it is possible to match the color characteristics of the selected pairs of devices. The problem of color management in such circumstances becomes more clear. Typically, color conversion occurs directly from the color gamut of one device to the gamut of another. Often, this conversion is established empirically.

In prepress and printing, there are two factors that make this approach less and less applicable. The first factor is the growth in the number of open systems (i.e. systems in which equipment from different manufacturers can be used and frequently reconfigured), which, actually, leads to the need to develop a concept of openness in color management. The second factor is the growth of distributed systems, in which document creation and reproduction often occur in locations many kilometers apart. This leads to the need for reliable coupling of individual parts of the system.

These problems can be solved by building a color management system based on a well-defined, neutral color-coding system, such as the color space CIE. If the

hardware-defined colors of a peripheral device can be translated into a hardware-independent color space, and if all computer and software manufacturers agree to use such a hardware-independent color space, it will become much easier to connect equipment from different manufacturers into a single system, as well as to maintain a single-color specification. Because the CIE color space is well-defined and reproducible, it is an excellent language for communicating color information between remote systems.

Therefore, in early 1993, several companies decided to join forces to develop a common approach to the problem of color management. They formed International Color Consortium (ICC) in order to solve user problems by creating a reliable color reproduction mechanism that ensures repeatable results at any stage of the reproduction process.

One of the first decisions made by ICC was to place the responsibility for color space conversion on the operating system, eliminating the need to embed a color management system into each application. On the other hand, each application can access the color management system. And device profiles, which contain information about the color behavior of various peripherals, provide the data necessary to perform color transformations.

Of course, ICC does not provide specific instructions on how to build an operating system and its architecture, but only general provisions. It is proposed to embed a Color Management Framework inside the operating system, which is responsible for the most important color management functions of the operating system, such as organizing profiles, supporting different color spaces, etc. This module provides an interface to various color conversion methods, which are the heart of a color management system, performing conversions of image color data into specific color spaces of output devices. The module supports both the CIE XYZ and CIE Lab color coordinate systems as standard. It is also possible to add other color coordinate systems. The possibility of using other systems is ensured by the well-defined specification and its openness. In addition, support is provided for color spaces of devices with different numbers of output channels. Profiles can be made for three channels (RGB, CMY, HSV), four channels (CMYK) or even for seven-color printing.

The ICC profile specification begins with a descriptive section that explains concepts such as device profiles, color spaces, profile docking space, profile structure elements, and embedded profiles. The content of device profiles is described from top to bottom. ICC profiles basically consist of a table of contents and the data that follows it. The document describing the profiles first defines the different types of device profiles and what tags should be in those profiles. Then comes the description of tags and their type. After that, the definition of the main types is given. The document contains examples and appendices. In particular, one of the most important appendices describes how to embed profile files in EPS, PICT and TIFF files. ICC defines different types of profiles:

– Input Device;

– Display Device;

– Output Device;
– Color Space Conversion;
– Device Linking;
– Abstract Profile.

A set of control tags is defined for each profile type. It is also possible to add any other tags.

One of the first steps in building a profile is to colorimetrically measure a set of color patches on some medium or display device. If the image medium or viewing environment differs from the reference one, it is necessary to adapt the measured colorimetric values, to bring them into line with the profile coupling space (PCS). Such adaptation is calculated based on the chromaticity of the white point, the relative luminance of an ideal reflector, viewing conditions, the viewing light source, and glare. Currently, the adaptation process is handled by the profile generation software.

For instance, when building a scanner profile, it is necessary to scan a reference image, the so-called color target, and match the resulting data to the reference data stored in the target's color measurement file. The colors contained in the reference image should be distributed as evenly as possible within the color space CIE. Comparison of reference data, i.e. colorimetric measurement data of the target supplied to the scanner, and data obtained during scanning of the reference image, provides good information about the reproduction properties of the scanner.

Printer profiling is the reverse process. In this case, a set of color patches evenly distributed over the printer's color gamut (CMY or CMYK model) is first generated, and then this set is printed. The printed patches are then colorimetrically measured. The conversion method that translates the CIE color space into the device's color gamut is more complex in this case.

Different manufacturers use different numbers of test color patches. Some vendors use the IT7.8 patch table, which contains 190 color patches, while other vendors use a test table of 4500 patches. Due to statistical noise, it is necessary to measure 15-20 samples and then average the results to obtain a reliable outcome. The question of how many samples should be measured to accurately characterize a device has not yet been resolved, despite its practical importance. This question is also unresolved because many color management systems are not so much aimed at the printing industry as at the desktop publishing market, and in this market segment it is unacceptable for a user to measure even 190 patches himself; these users would like to have profiles already built. Of course, this raises an obvious problem: the cost of such profiles intended for device characterization must be acceptable.

In a workflow that uses digital image processing, this data is combined with a profile that characterizes the device itself. When an image is required on an output device, the color data is converted, using input and output profiles, into the form in which it will be presented to the output device.

The workflow built around the ICC standard uses a different data processing model. There are three reasons for this. First, the color management system must support a wide variety of devices. Second, an image may be created in one location

and displayed in another, geographically distant location. Finally, the same image must be reproduced in different media (e.g., printed, film, and video) using the same tools and processes.

For these reasons, colors cannot be adjusted to the output device immediately after scanning, for example. They cannot be adjusted at any other point in the image creation process. Color management only occurs at the output device. This means that at the time an image is created, it is necessary to know on which output device it will be rendered. But just as the creator of an image does not know on which device it will be rendered, the person rendering the image may not know the color characteristics of the device on which the image was created.

The solution to this problem can be divided into two parts. When a document is created, an ICC profile is embedded in it, which maps the device's color gamut to a well-known color space. When a document is output, the operating system uses input and output profiles to convert data from the source color gamut to the output device color gamut. Actually, the problem can be solved slightly better by applying only one transformation without converting the input image to the CIE color space. There are two reasons for this. First, in real-world production environments, the source image is often processed on the same monitor using several different software tools. In this case, converting the image to CIE color space and back is a waste of time. Second, there is some loss of color accuracy due to rounding errors when performing color space conversions. In order to minimize data manipulation, it is possible to construct device color space conversions directly between each other. Unlike earlier conversions of this type, the use of ICC profiles allows any devices to connect to each other, and the conversions themselves can be performed by any operating system.

In order to use ICC profiles in a color management workflow, minimal changes are required. If there is a set of equipment consisting of scanners, monitors, various output devices and software, then the actual workflow can be represented by the following operations:

– scanner characterization using a profile building tool;

– monitor characterization using a profile building tool;

– characterization of output devices using a profile building tool;

– scanning and opening the image with a software tool like Photoshop;

– converting a scanned image into the color space of a monitor or other output device;

– inserting images into other programs, and, if necessary, performing further conversion to the color space of the monitor or other output devices;

– output on the output device.

As can be seen from this scenario, the use of profiles, which enables the color management system's mathematical framework to perform color space conversions between different devices for image rendering, constitutes the essence of the described approach.

Purpose and objectives of the study

As is known, the profile itself does not change the numerical RGB or CMYK values – it simply assigns them a specific meaning, essentially indicating that this set of RGB or CMYK numerical values represents a specific color defined in the Lab color space. Since color coordinate systems are three-dimensional spaces, a profile cannot be defined using conventional algebraic functions but instead represents a one-to-one correspondence between sets of color coordinates, specified in tabular form.

The table must contain function values for all combinations of color coordinates to ensure accurate calculations. For example, in the RGB model with 8-bit color depth, this would require 2563 = 16 777 216 points, which would, accordingly, require measuring the same number of color samples and nearly 50 MB of memory to store the table. If there are not three but four color channels (when working with data in the CMYK color model), the table size may increase to almost 13 GB. Therefore, in practice, tables usually contain no more than a few thousand points, and the missing values are calculated by interpolation based on the available points in the table.

However, even these few thousand points cannot be practically measured. Color targets do not contain such a number of measurement fields, and even if they did, measuring all of them would become a very labor-intensive and unjustified process. Therefore, during profile creation, there arises a need to approximate the measured data in order to obtain the missing points for the tables.

There are specialized software tools that allow performing such operations. However, all of them have their own drawbacks. The aim of this work is to address these issues:

– to investigate the possibility of carrying out such an approximation using artificial neural networks;

– compare the proposed developed method with existing ones.

Main part

In this study, artificial neural networks were used to approximate experimental points during the construction of an input profile.

The experiment was conducted using the MATLAB R2006b mathematical software package. MATLAB is a high-performance language for engineering and scientific computing. It supports mathematical calculations, scientific graphics visualization, and programming within an interactive environment that is easy to learn, especially when tasks and their solutions can be expressed in a notation close to mathematical form.

The most well-known application areas of MATLAB include mathematics and computations; algorithm development; computational experiments, simulation modeling, prototyping; data analysis, exploration, and visualization of results; scientific and engineering graphics; and application development, including the creation of graphical user interfaces.

The Neural Network Toolbox extension in MATLAB provides tools for designing, modeling, training, and utilizing a wide range of well-known artificial neural network paradigms, from basic perceptron models to the latest associative and self-organizing networks. The toolbox can be used to solve a variety of tasks, such as signal processing, nonlinear control, financial modeling, and more.

For each type of architecture and training ANN algorithm, there are functions for initialization, training, adaptation, creation, simulation, demonstration, and application examples. Artificial multilayer neural networks are constructed based on the principles of their biological counterparts. They are already capable of solving a wide range of tasks such as pattern recognition, identification, control of complex nonlinear objects, robotics, and more.

As the output data for solving the problem, a second-order polynomial was used, with an added random component ε, normally distributed with zero expected value and a variance of σ^2. The polynomial has the following form:

$$Y = a_0 + a_1 x + a_2 x^2 + \varepsilon, \quad (1)$$

where $Y = \{Y_i\}_{i=0}^{n}$ – output data vector;

$x = \{i * 0.01\}_{i=0}^{n}$ – input data vector; n – sample size;

$a = (a_0, a_1, a_2)$ – constant coefficients vector.

Output data generation was performed at values of n = 1000, a_0 = 2; a_1, a_2 = 1; σ = 4.

From this sample, 100 points were randomly selected using the rand function and used to train the network. For the reproducibility of the experiment, the state = 20 parameter of this function was fixed, so that the same points would always be selected when the program was run. To test the network performance based on the rand function, another sample of 100 values was taken. The state parameter value was 10. It should also be noted that points that were the same for these two samples were excluded.

The approximation was performed using three functions of the package Neural Network Toolbox: newgrnn, newrb, newrbe.

The newgrnn function implements the creation of a generalized regression neural network. It designs a two-layer network where the first layer, consisting of radial basis elements, calculates the input weights using the Euclidean distance weighting function, and the network outputs using the input multiplication function. The second layer contains neurons that apply a linear activation function. It computes the input weights based on the normalized dot product, and the network outputs are returned using the input summation function. Only the first layer has biases.

Newgrnn assigns the input weights to the first layer, with all biases set to a value equal to 0.8326/spread, where spread is the spread parameter of the function. The second layer assigns weights according to the target output values.

Figure 1 shows the approximation of experimental data using the newgrnn function (smoothing parameter = 0.5).

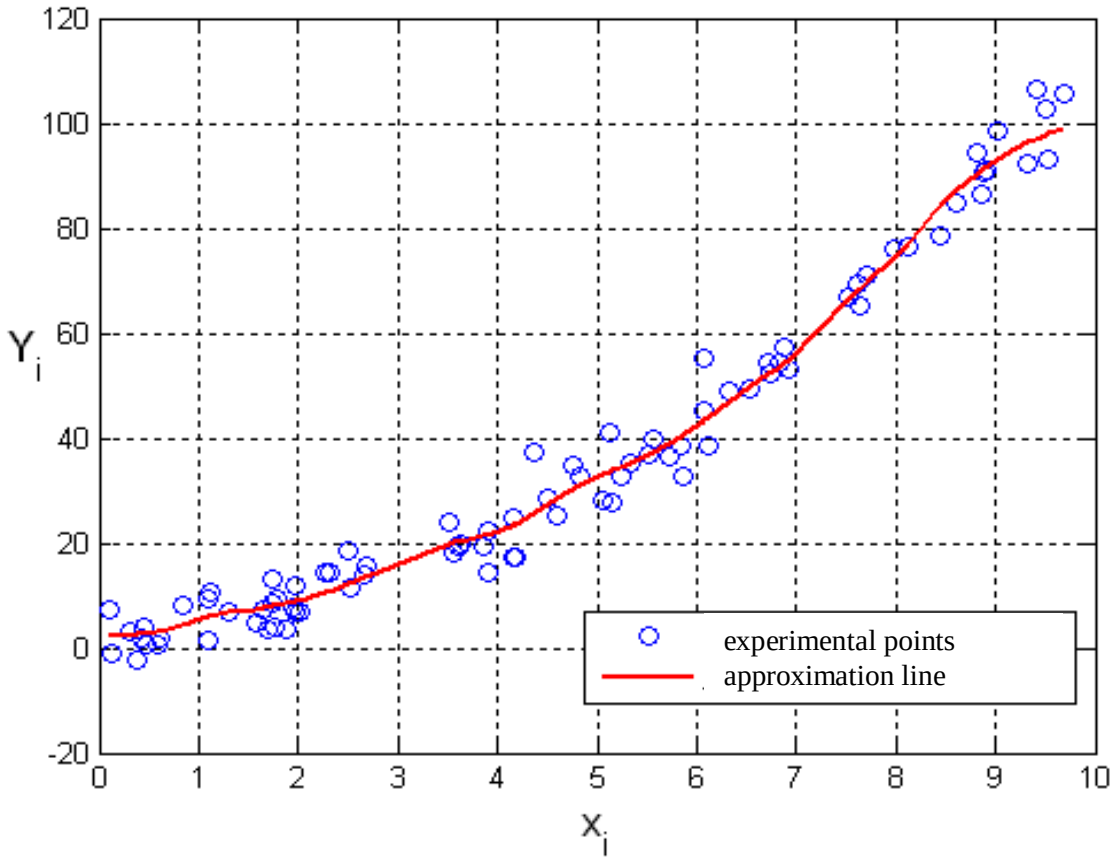

Figure 1 – Approximation using a function newgrnn

The newrbe function constructs an exact radial basis function (RBF) network. The created two-layer network uses a weight assignment principle similar to the network designed with the newgrnn function. The difference is that for weight assignment in the second layer, a dot product weighting function is used. Also, in this network, both layers can have biases assigned.

Figure 2 illustrates the approximation of the output data by a network built using the newrbe function (smoothing parameter = 8).

The newrb function implements the construction of a radial basis network. It is fully analogous to the newrbe function but includes an additional parameter – goal, which sets the threshold value for the training criterion. Initially, the radial layer of the network contains no neurons. Neurons are added during the network construction process until the squared deviation reaches the value specified by the goal parameter. Until this value is achieved, the following steps are repeated during the network construction process:

– network modeling;

– finding the input vector with the largest error;

– adding to the first layer a neuron with weights equal to this vector;

– redefining the weights of the second layer in such a way as to minimize the error.

The approximation based on the use of the newrb function is depicted in fig. 3 (smoothing parameter = 8).

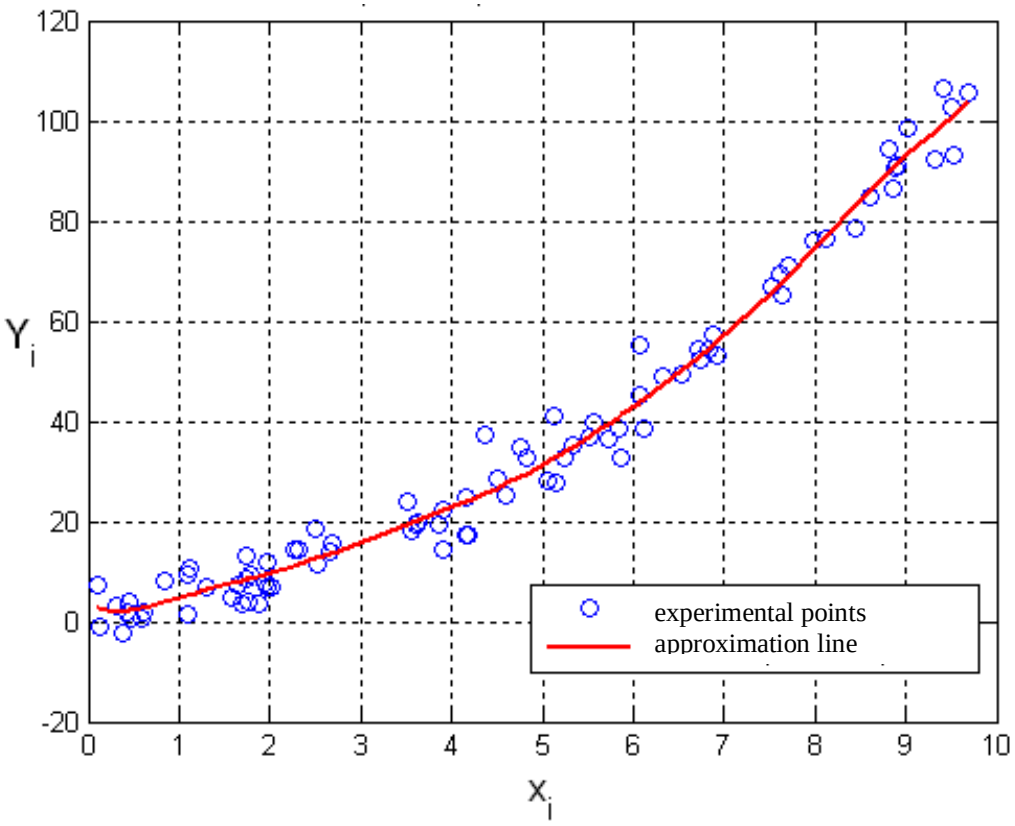

Figure 2 – Approximation using a function newrbe

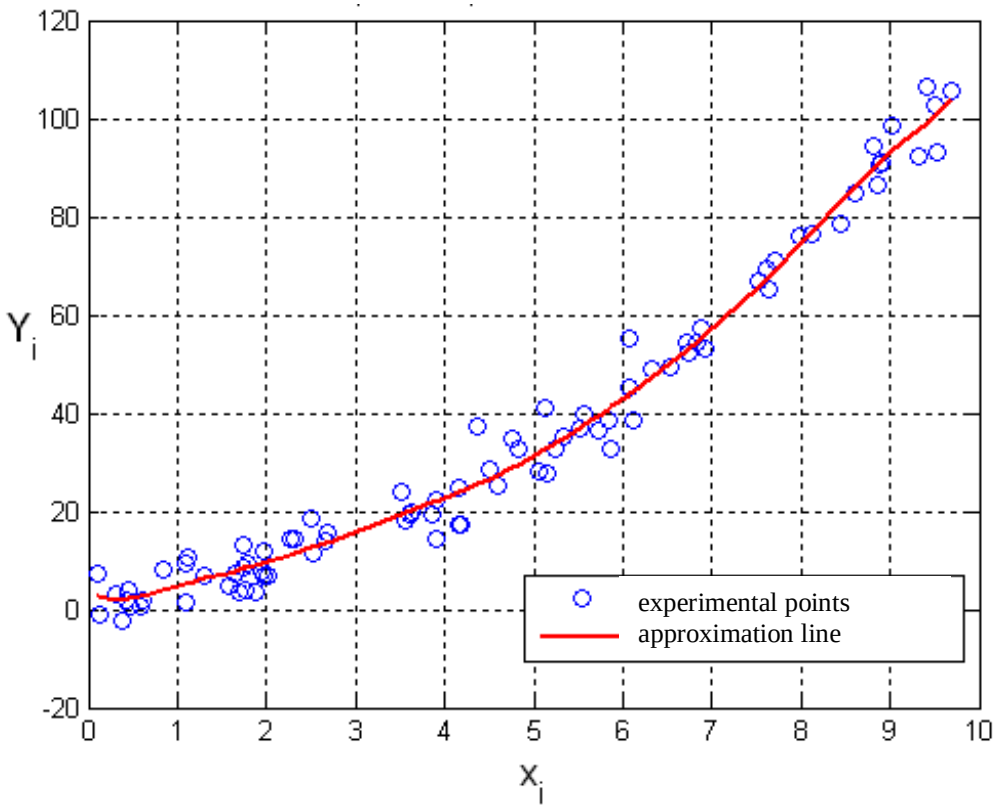

Figure 3 – Approximation using a function newrb

As the criterion for evaluating the use of Neural Network Toolbox functions, the squared deviation of experimental points from the approximation line was chosen. Ideally, this value should approach zero.

For each of the functions used, the dependence of the squared deviation on the smoothing parameter was studied. The results are presented in the form of graphs (fig. 4-5). It is important to note that for each function, the range of the smoothing parameter was selected to be the most optimal for use. Thus, for the newgrnn function, the most acceptable interval was [0.1, 1.5], and for newrbe and newrb – [5.9, 9.5].

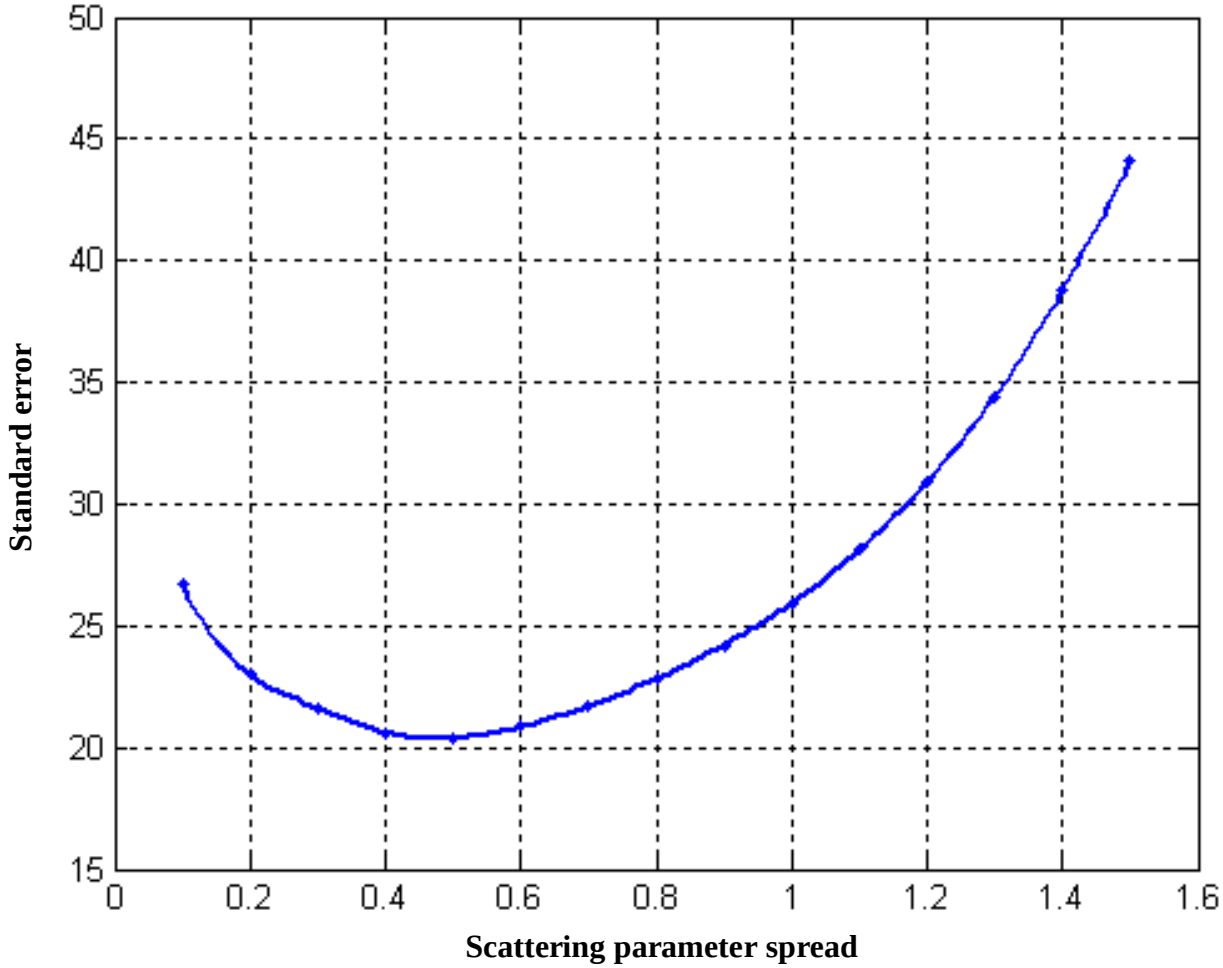

Figure 4 – Dependence of the squared deviation on the smoothing parameter when using the function newgrnn

When using the newgrnn function, the minimum value of the squared deviation corresponds to a spread of 0.5. spread = 0.5. Further on the graph, a monotonically increasing dependence can be observed, meaning that as the smoothing parameter increases, the network error also increases. This indicates that the network begins to retrain – it starts approximating the available data too closely.

During the study of the newrb function, it was found that the dependence line has an uneven character. In the range close to spread = 9, a sudden jump is observed: initially, the squared deviation becomes minimal at spread = 8.85, and then the error value increases by several tenths before the graph stabilizes again. The problematic area was investigated with a finer step, and it was found that there is no algorithm convergence within this interval.

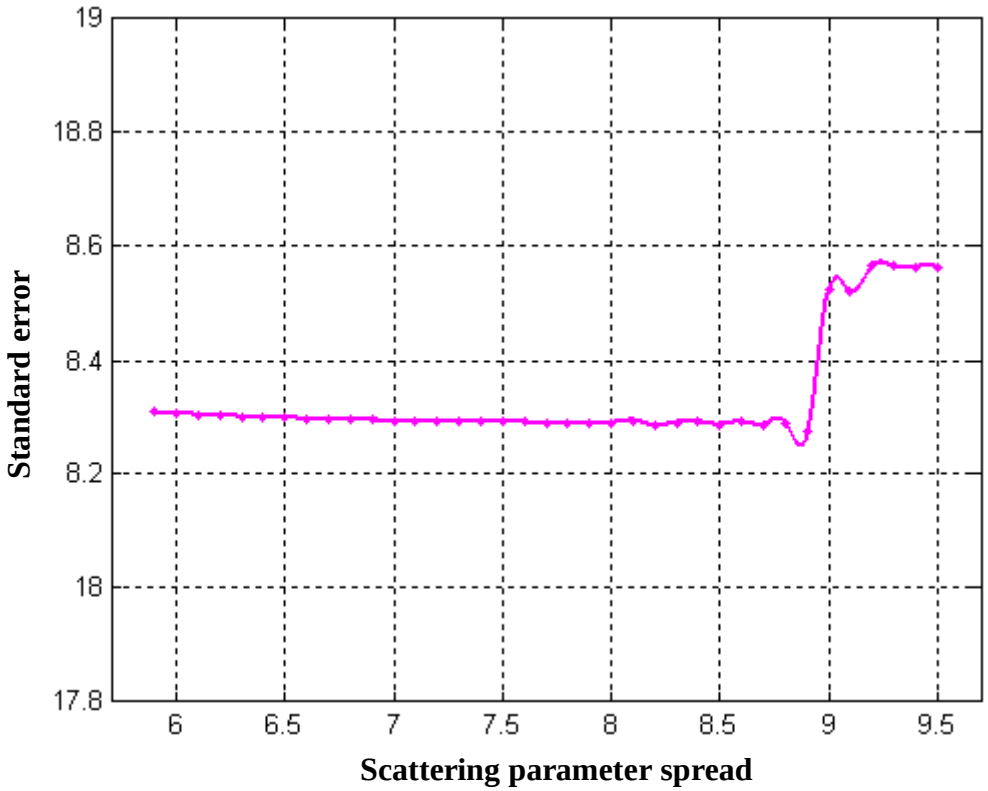

Figure 5 – Dependence of the squared deviation on the smoothing parameter when using the function newgrnn

The dependence of the squared deviation on the smoothing parameter when using the newrbe function practically coincides with the values obtained for the newrb function.

The Neural Network Toolbox package provides the capability to create a "classic" multilayer neural network using the newff function. In multilayer networks, neurons are grouped into layers. A layer consists of a collection of neurons that share the same input signals. The number of neurons in each layer can be arbitrary and is not predetermined by the number of neurons in other layers. In general, the network consists of Q layers, numbered from left to right. External input signals are fed to the input neurons of the first layer (the input layer is often numbered as zero), and the outputs of the network are the output signals of the last layer. The network input can be considered as the output of a "zero layer" consisting of degenerate neurons that serve only as distribution points, where no signal summation or transformation occurs. Besides the input and output layers, a multilayer neural network has one or more intermediate (hidden) layers. The connections from the outputs of the neurons in a given layer q to the inputs of neurons in the next layer (q+1) are called sequential connections.

In the newff function, the training of such a network is performed using the back propagation method. This is an iterative gradient-based training algorithm used to minimize the squared deviation between the current output and the desired output in multilayer neural networks. In this algorithm, the error function is defined as the sum of squared discrepancies (errors) between the desired and actual outputs of the network.

When calculating the elements of the gradient vector, a specific form of the derivative of the sigmoid activation functions is used. The algorithm operates cyclically, and these cycles are conventionally called epochs. During each epoch, all training samples are fed sequentially to the network input, the network's output values are compared to the target values, and the error is computed. The error value, along with the gradient of the error surface, is used to adjust the weights, after which all actions are repeated. The initial configuration of the network is chosen randomly, and the training process is stopped either when a set number of epochs is completed, when the error reaches a predefined low level, or when the error ceases to decrease (the user can select the desired stopping condition).

The classical back propagation method belongs to algorithms with linear convergence. Its well-known drawbacks include: relatively low convergence speed (a large number of iterations needed to reach the minimum of the error function), possibility of converging to local solutions (local minima of the error function rather than the global minimum). A network paralysis is also possible, when most neurons operate at very large argument values of the activation function, i.e., on its flat region (since the error is proportional to the derivative, and the derivative is small in these regions, the training process virtually stagnates).

To eliminate these drawbacks, numerous modifications of the back propagation algorithm have been proposed, involving the use of various error functions, different procedures for determining the direction and magnitude of the learning step, and so forth.

The network used for this work applies the most efficient variant of the backpropagation algorithm, namely the Levenberg-Marquardt method, which assumes that the function modeled by the neural network is linear. Under this assumption, the minimum can be determined in a single computation step. The found minimum is then verified, and if the error has decreased, the weights are updated with the new values. The entire procedure is repeated sequentially. Since the assumption of linearity is generally not fully justified, it may happen that one would need to check points lying far (sometimes very far) from the current point. In the Levenberg-Marquardt method, the location of the new point is a compromise between moving in the direction of steepest descent and making a large jump based on the linearity assumption. Successful steps are accepted, shifting the balance towards the linearity assumption (which is approximately valid near the minimum point), while unsuccessful steps are rejected, and the algorithm moves more cautiously downhill. Consequently, the Levenberg-Marquardt algorithm constantly adapts its behavior and can operate very quickly.

To make rational use of the network's ability to find the minimum mean squared error, an approach implemented in the Group Method of Data Handling (GMDH) algorithms is used. In this method, all optimization issues are solved by selecting options based on the available training and validation data sequences.

For the experiment, the complex nftool function from the Neural Network Toolbox package was used (fig. 6). This application guides the user through solving the task in dialog mode, implementing it through a feedforward network trained using

the Levenberg-Marquardt method. Nftool requires loading data into the MATLAB workspace. This is done using the load command, which imports data from the specified file into MATLAB's workspace, making them ready for processing. Next, it is necessary to set the number of neurons in the hidden layer and define the percentage split between the training and validation sequences. Data normalization, sequence splitting, network creation, training, and result visualization are all performed automatically by the application.

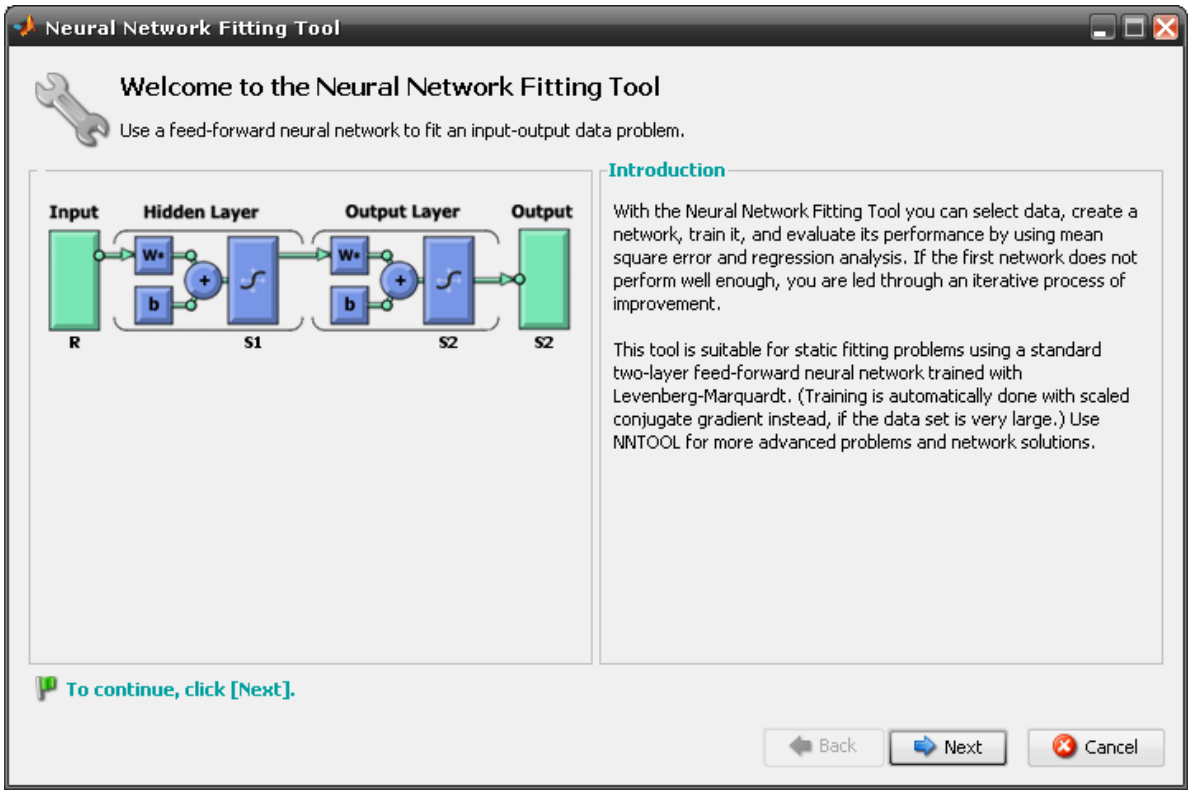

Figure 6 – Nftool program dialog box nftool

The output data used for the experiment consist of samples of input (RGB) and output (Lab) measured color values. These data were obtained by measuring a color target Eye One™ Scan Target 1.4 with Eye One Pro spectrophotometers.

The data is presented in the form of matrices of dimension 3×288. However, the network uses only one output element in the work. Therefore, the experiment was performed three times for each row of the Lab matrix with the three input elements (RGB matrix) unchanged.

Initially, the input and output values are normalized to the range [−1, 1]. The normalization information is stored within the network to enable inverse transformation later. Then, the output data is split into three sequences: for training, validation, and testing. After that, the network is created and trained, and a reverse transition to denormalized data is performed, with results presented in the form of graphs. It is important to note that the initial weight values assigned to neurons depend on the initial point, which is chosen randomly. To ensure experiment reproducibility, this point was fixed. Thus, calculations are performed with the same initial weight values, making it possible to change all the other parameters.

After building a neural network using nftool for the output L*, with the number of neurons in the hidden layer set to 20 and both the training and validation sequences comprising 20% each, the results were obtained in the form of two graphs shown below. The first graph illustrates the dependence of the squared deviation on the number of iterations for each sequence (fig. 7). It is evident that with each subsequent epoch, the network error decreases, approaching the default target value goal = 0. As can be seen from the graph, it took 14 iterations to reach a deviation equal to 0.00450785.

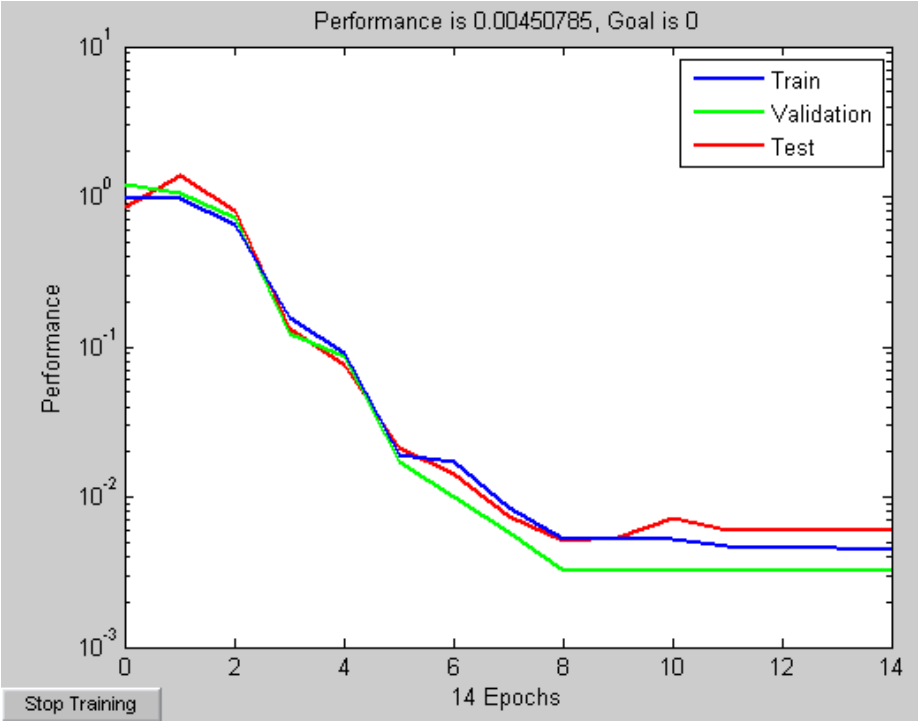

Figure 7 – Dependence of the mean square error on the number of iterations during network training using nftool

The second graph displays the dependencies between the network outputs and the experimental data, including the value of the multiple coefficients of determination R-squared. (fig. 8).

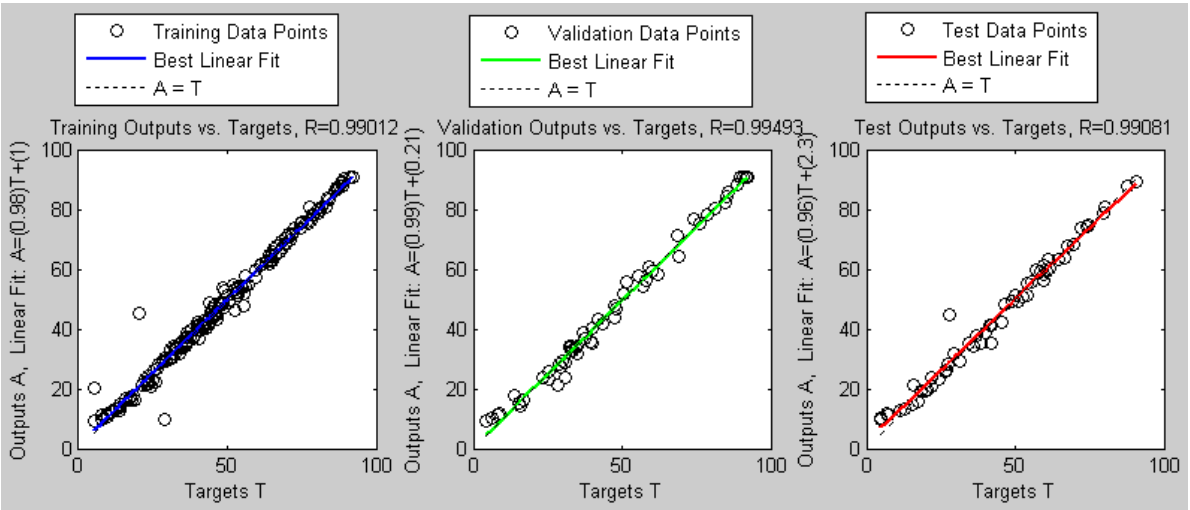

Figure 8 – Dependences of network outputs on experimental data

The coordinates of the points are given relative to the output data. The Linear Fit field displays the equation of the approximation line for each sample. In the case of perfect approximation, all points lie on a single line, which corresponds to the bisector of the angle.

Several methods for splitting available data into training and validation sets have been proposed online. These methods are based on calculating the squared deviations from the mean value for all available interpolation nodes.

The first method is aimed at obtaining such a division of nodes that ensures the greatest conditioning of the normal equation matrices. To achieve this, the points are ranked according to the value of D^2, with those exhibiting greater variance assigned to the training sequence, and those with lower variance assigned to the validation sequence. The ratio of the number of points in both sequences is further refined by evaluating several alternatives based on the criterion of minimizing the mean squared error, which is calculated on the second validation sequence.

The objective of the second method is to achieve an approximate equality of the statistical characteristics between the validation and training sequences (that is, the equality of means, variances etc.). To ensure statistical equivalence, the interpolation nodes are ranked according to the value of the variance D^2 and numbered accordingly. Subsequently, the nodes with even indices are used to form one sequence, while those with odd indices form the other. For different ratios of sequence sizes, it is possible to select all points with indices divisible by three, four, and so on, for inclusion in the validation sequence. As in the previous method, the ratio of the sequence sizes can be determined through an optimization process.

According to these recommendations, the original sample was divided into training and testing sequences using the first method. The results obtained in this case are presented in figures 9-10.

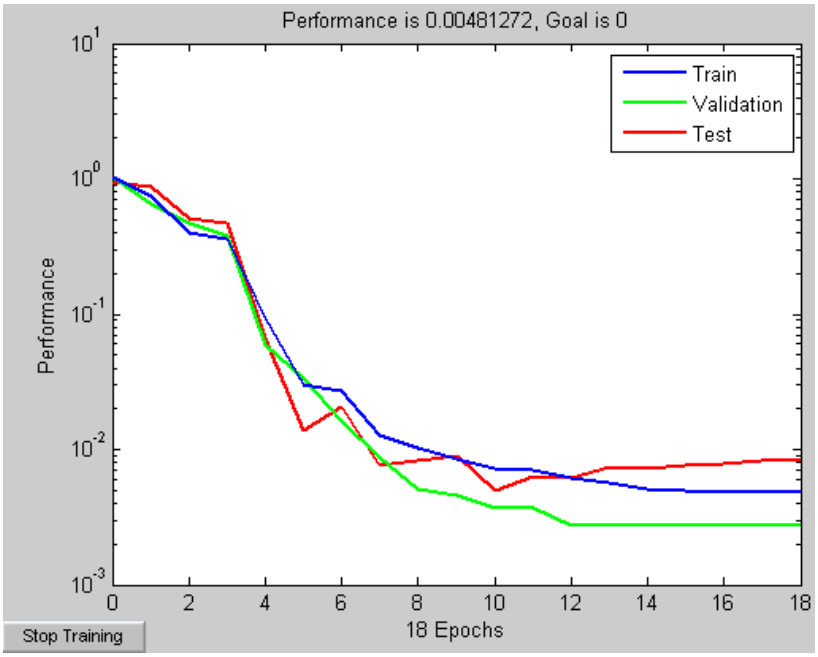

Figure 9 – Dependence of the mean square error on the number of iterations when dividing the sample into training and testing sequences after ranking by increasing variance

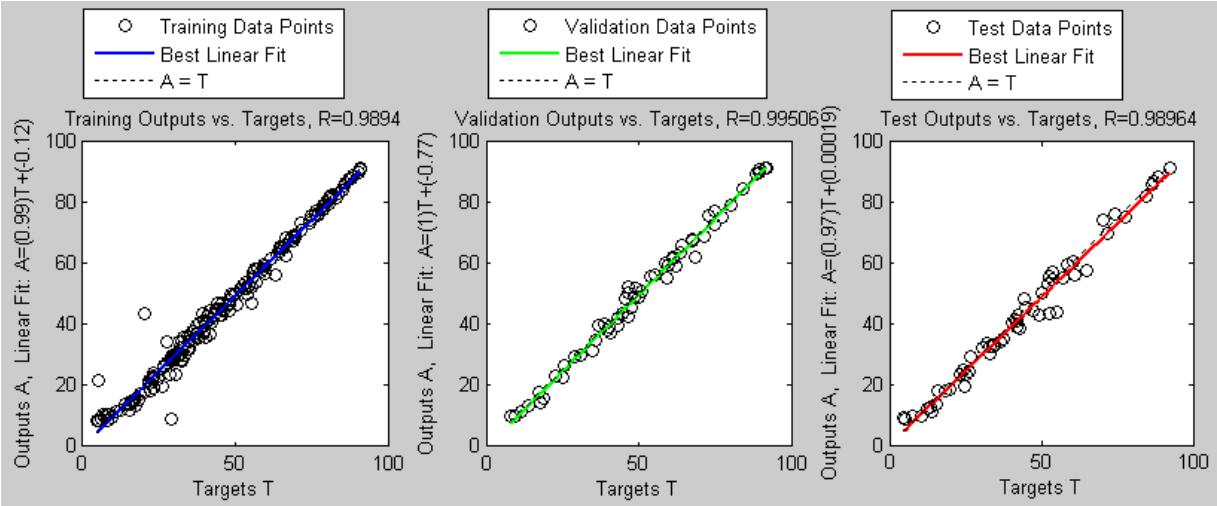

Figure 10 – Dependences of network outputs on experimental data in the case of dividing the sample into training and testing sequences after ranking by increasing variance

The graphs show that although the network required a slightly larger number of iterations than in the first case, there is no significant difference – the network error is only a few ten-thousandth higher than the previous value, and the R-squared value is also close to unity.

To validate these results and to potentially identify the optimal ratio between the training, testing, and validation sequences, several additional graphs were constructed, in which the target output values and the percentage distribution between the sequences were varied. The results are presented in the figures 11-14.

The graphs located on the left and at the top correspond to the outcomes obtained exclusively through the use of nftool, while those on the right and at the bottom correspond to the division of the sequences according to the same first method.

Also, for the first case, a network was built in which the initial data for division into a validation and training sequence was ranked in decreasing order, that is, those points with a larger variance were assigned to the test sequence, and the smaller one to the training sequence. The results of such an experiment are presented in fig. 15-16.

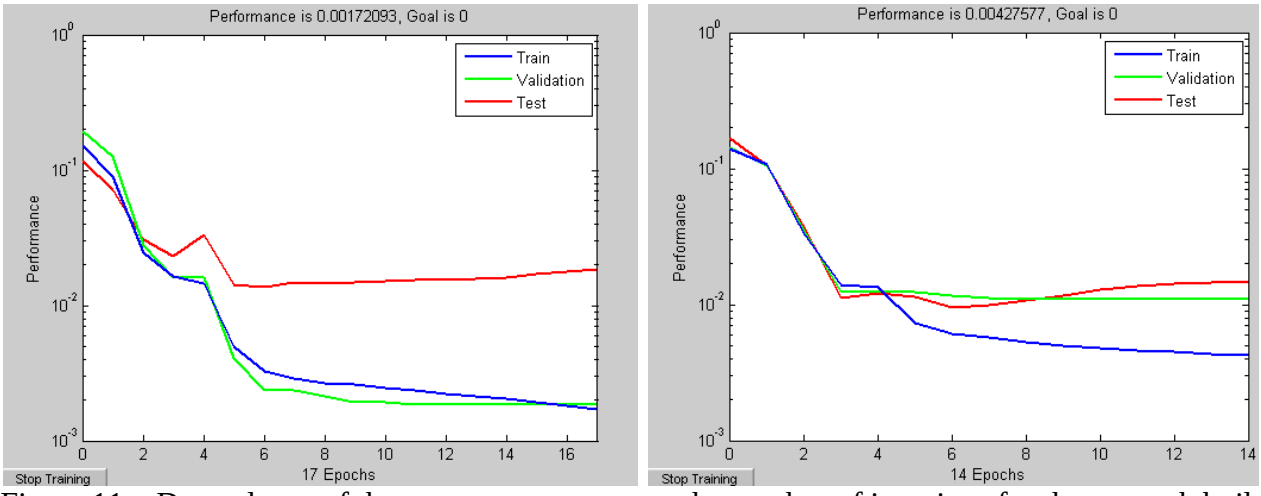

Figure 11 – Dependence of the mean square error on the number of iterations for the network built for the output a*, for a ratio of training and testing sequences of 20% and 30%, respectively

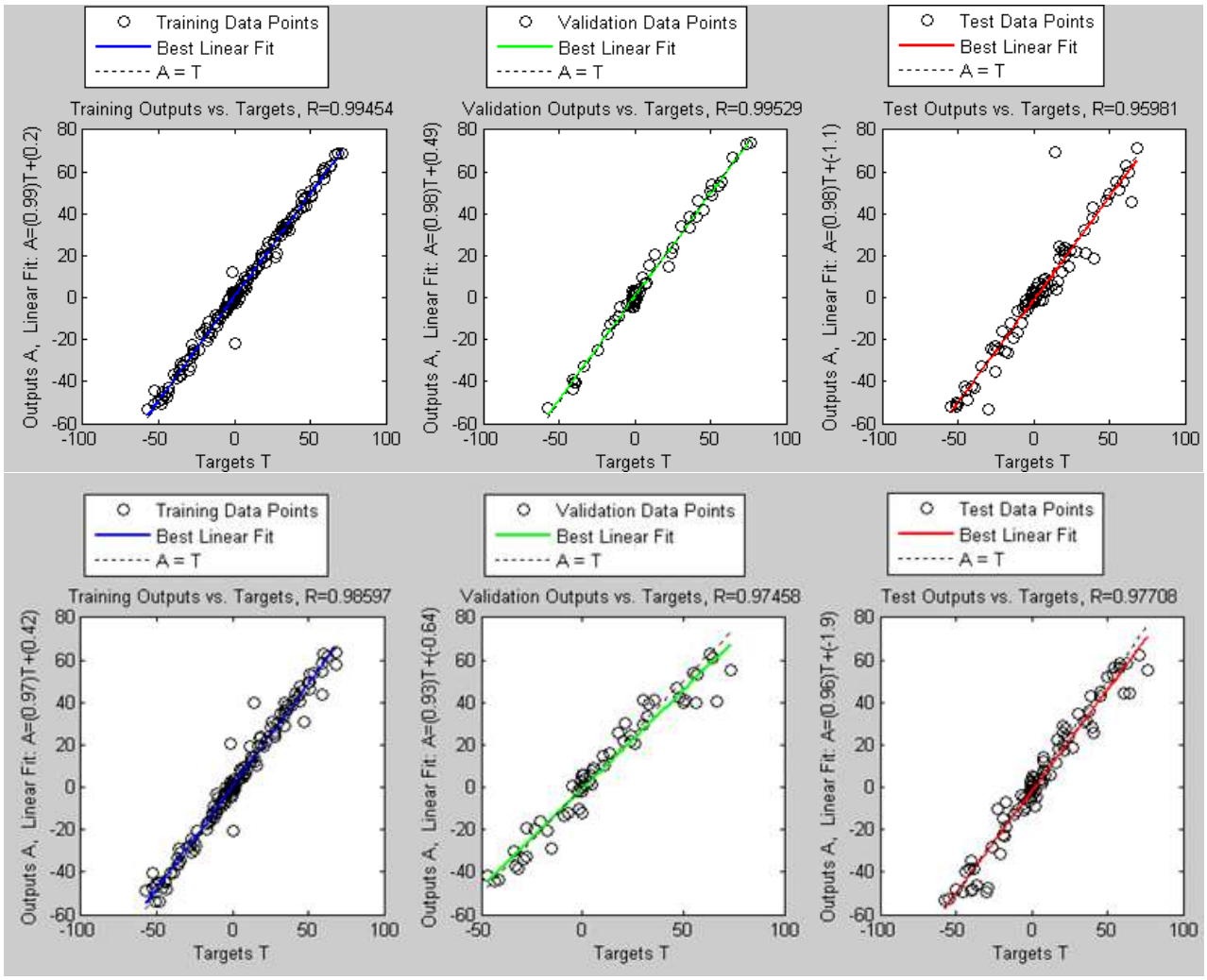

Figure 12 – Dependences of network outputs on experimental data for a network built for output a*, with a ratio of training and testing sequences of 20% and 30%, respectively

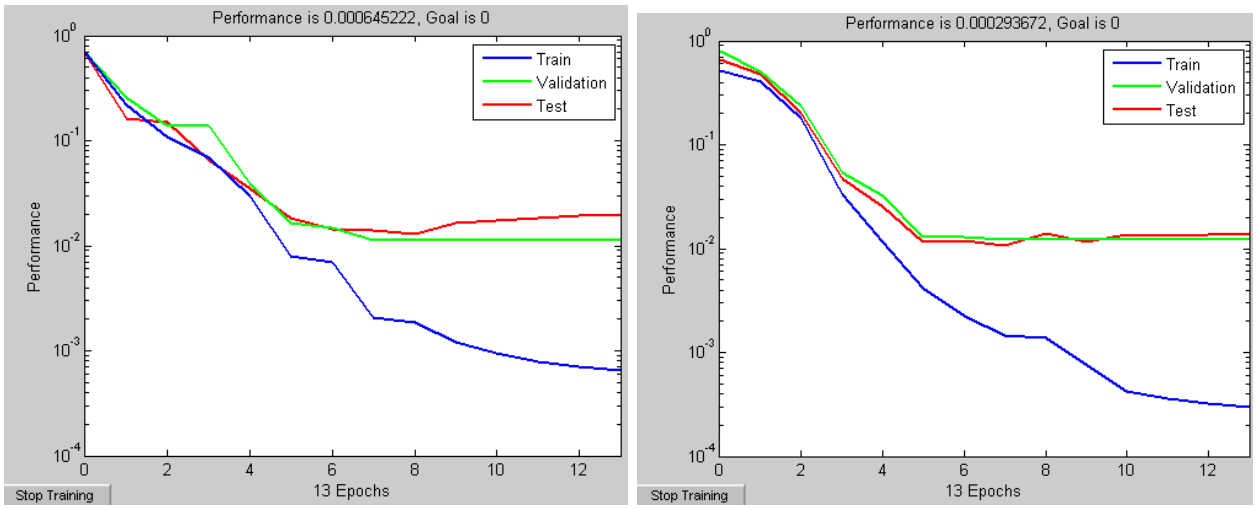

Figure 13 – Dependence of the mean square error on the number of iterations for the network built for the output b*, for a ratio of training and testing sequences of 40% i 30%, respectively

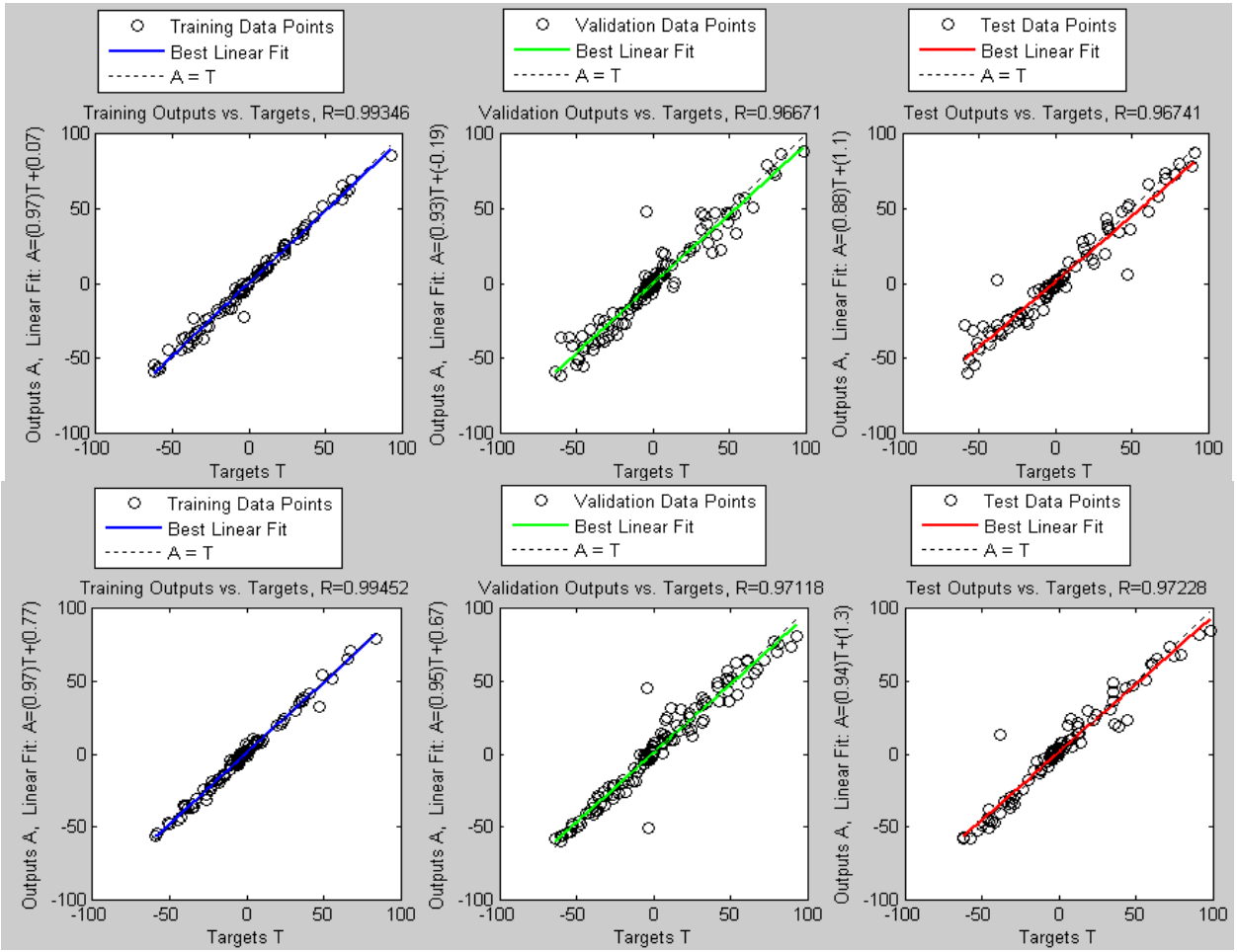

Figure 14 – Dependences of network outputs on experimental data for a network built for output b*, with a ratio of training and testing sequences of 40% i 30%, respectively

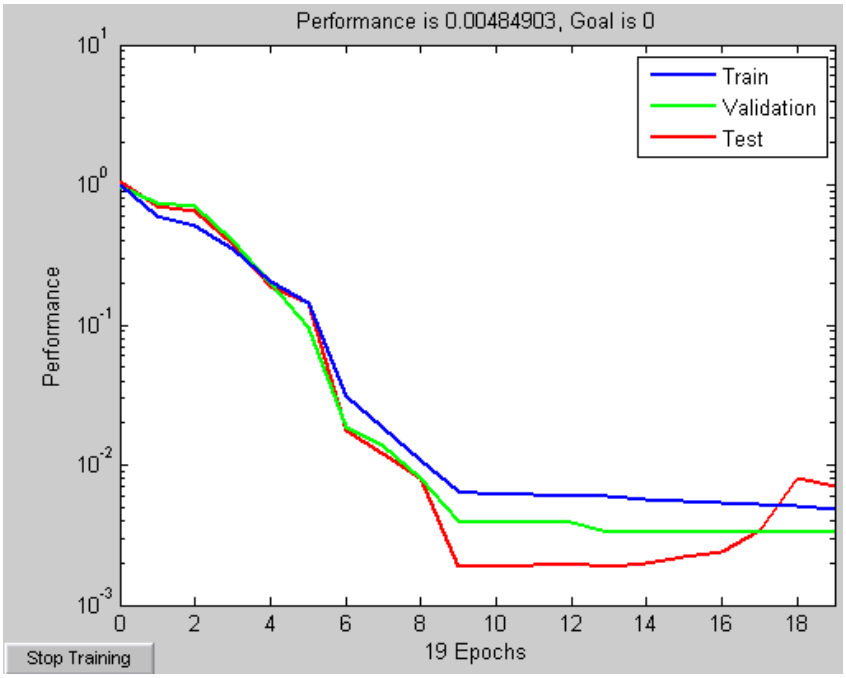

Figure 15 – Dependence of the mean square error on the number of iterations when dividing the sample into training and testing sequences after ranking by variance reduction

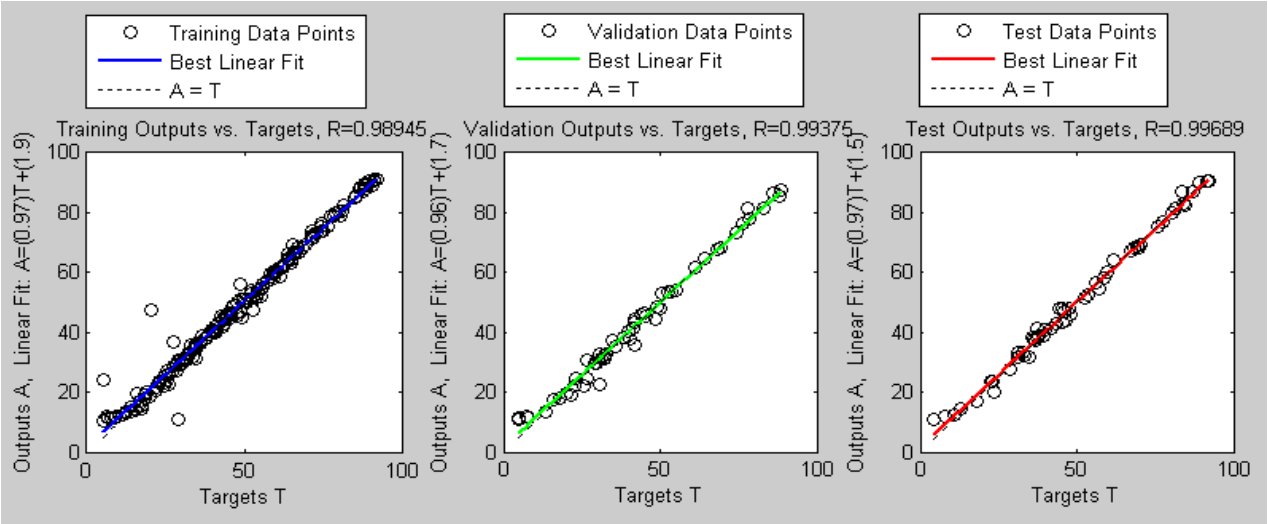

Figure 16 – Dependences of network outputs on experimental data in the case of dividing the sample into training and testing sequences after ranking by variance reduction

Thus, it is evident that the approach involving the partitioning of data into training and validation sequences according to the method proposed in the recommendations for the Levenberg–Marquardt algorithm does not yield a significant improvement in solving the approximation problem using artificial neural networks.

Additionally, a study was conducted to investigate the effect of the number of neurons in the hidden layer on the network's performance. It was observed that as the number of neurons increased, the mean squared error of the network for the same output data decreased. However, no clear dependency was found between the number of epochs required for training the network and the number of neurons in the hidden layer. The results of the study are presented in figures 17-22.

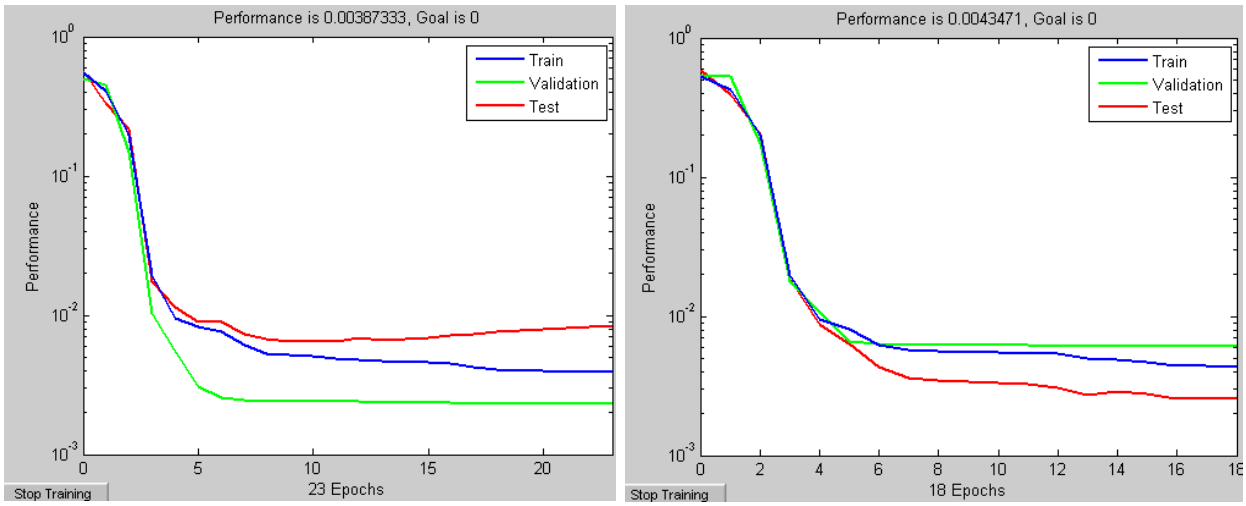

Figure 17 – Dependence of the mean square error on the number of iterations for the network built for the output L*; number of neurons in the hidden layer = 10

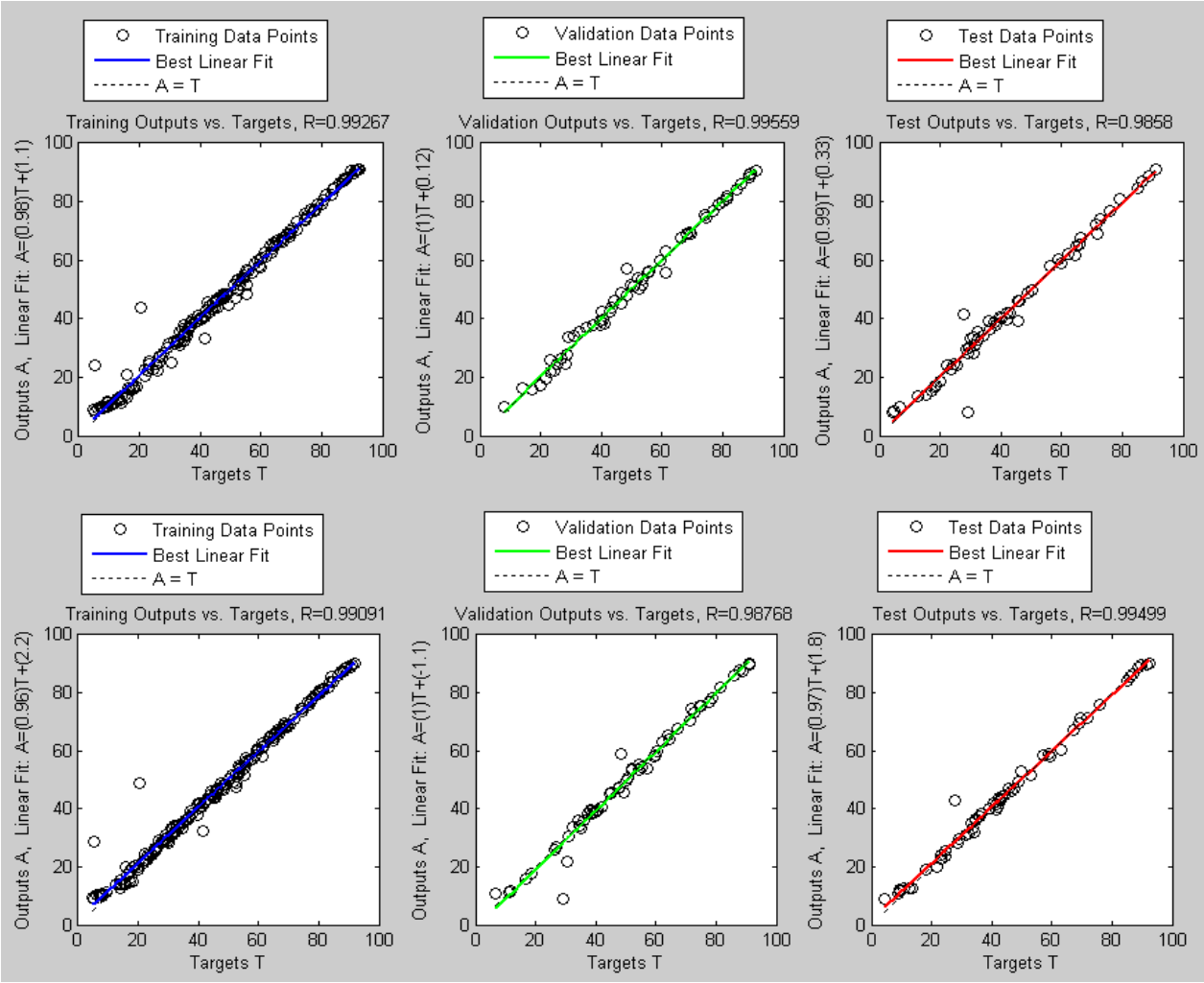

Figure 18 – Dependences of network outputs on experimental data
for a network built for output L*; number of neurons in the hidden layer = 10

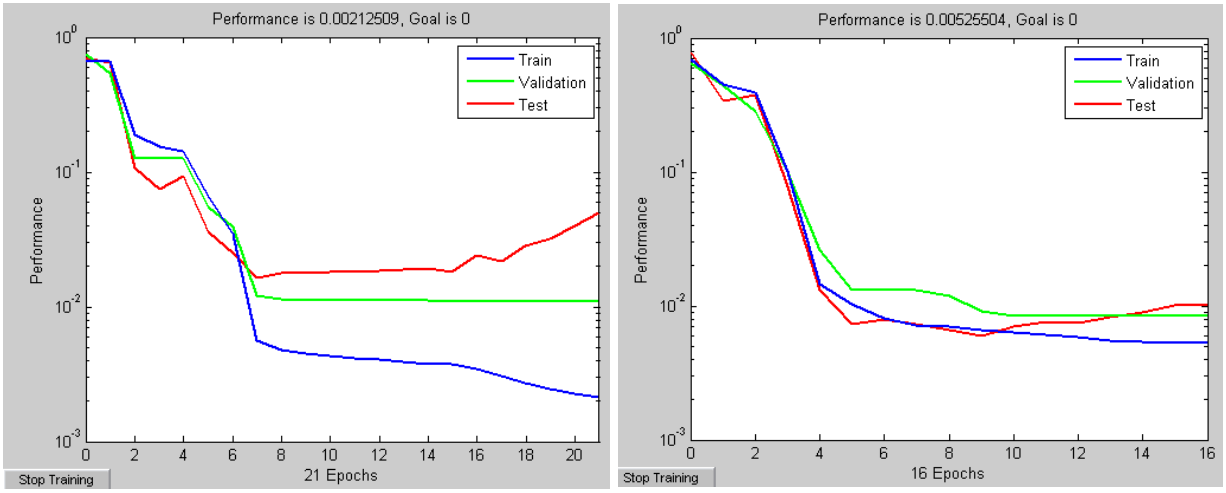

Figure 19 – Dependence of the mean square error on the number of iterations for the network for the output a*; number of neurons in the hidden layer = 25

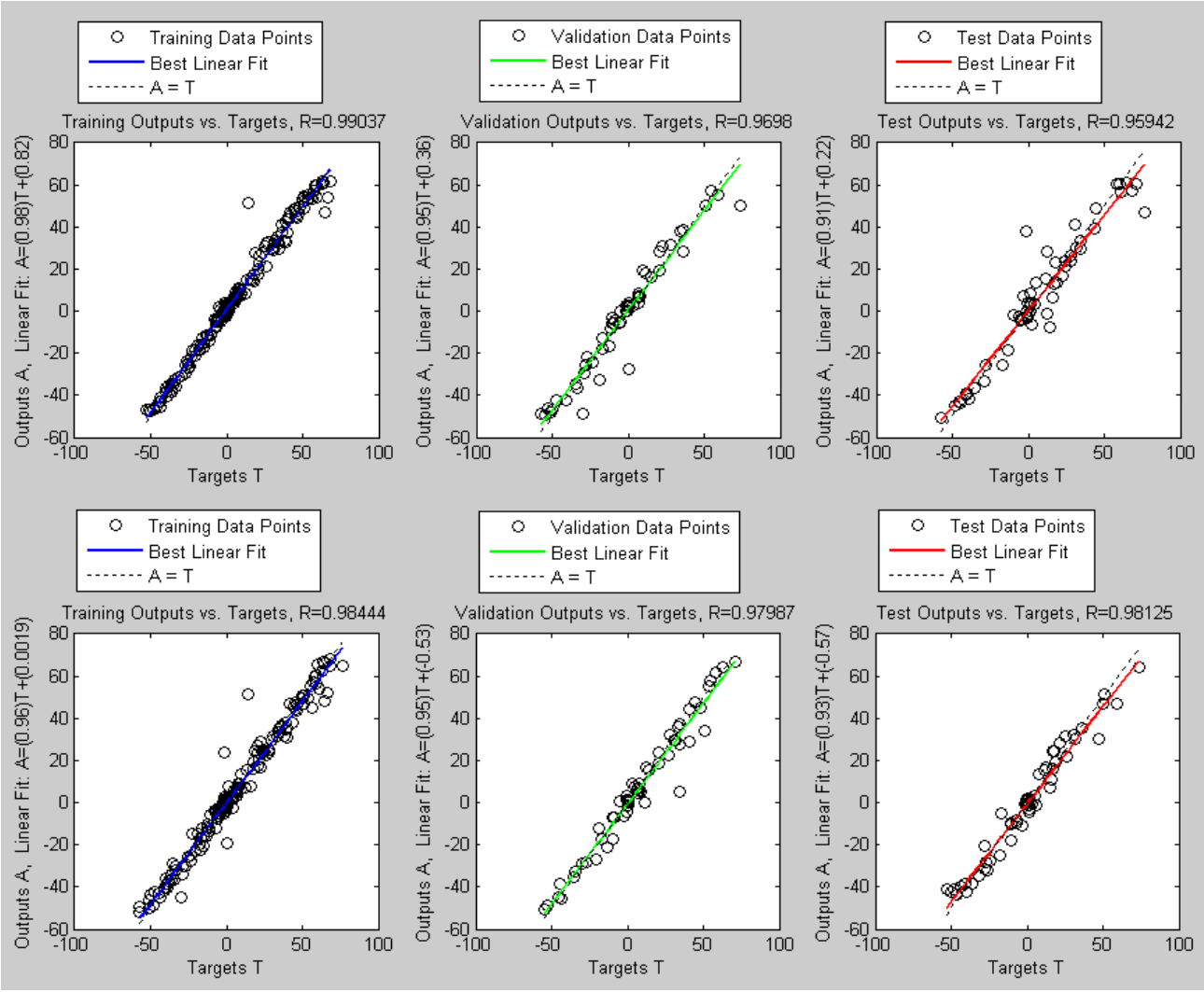

Figure 20 – Dependences of network outputs on experimental data for a network built for output a*; number of neurons in the hidden layer = 25

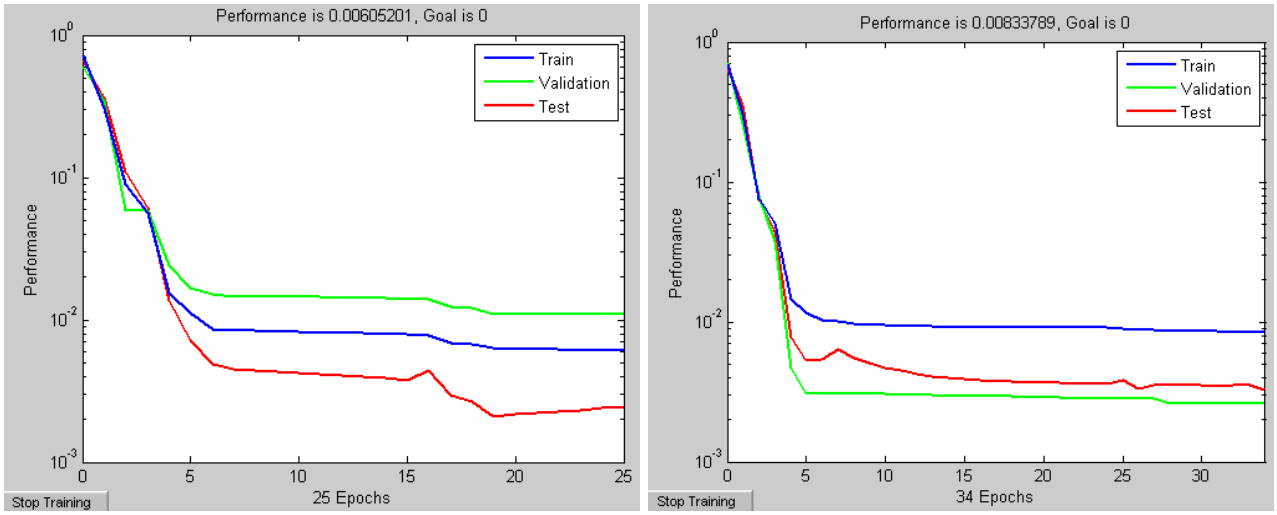

Figure 21 – Dependence of the mean square error on the number of iterations for the network for output b*; number of neurons in the hidden layer = 5

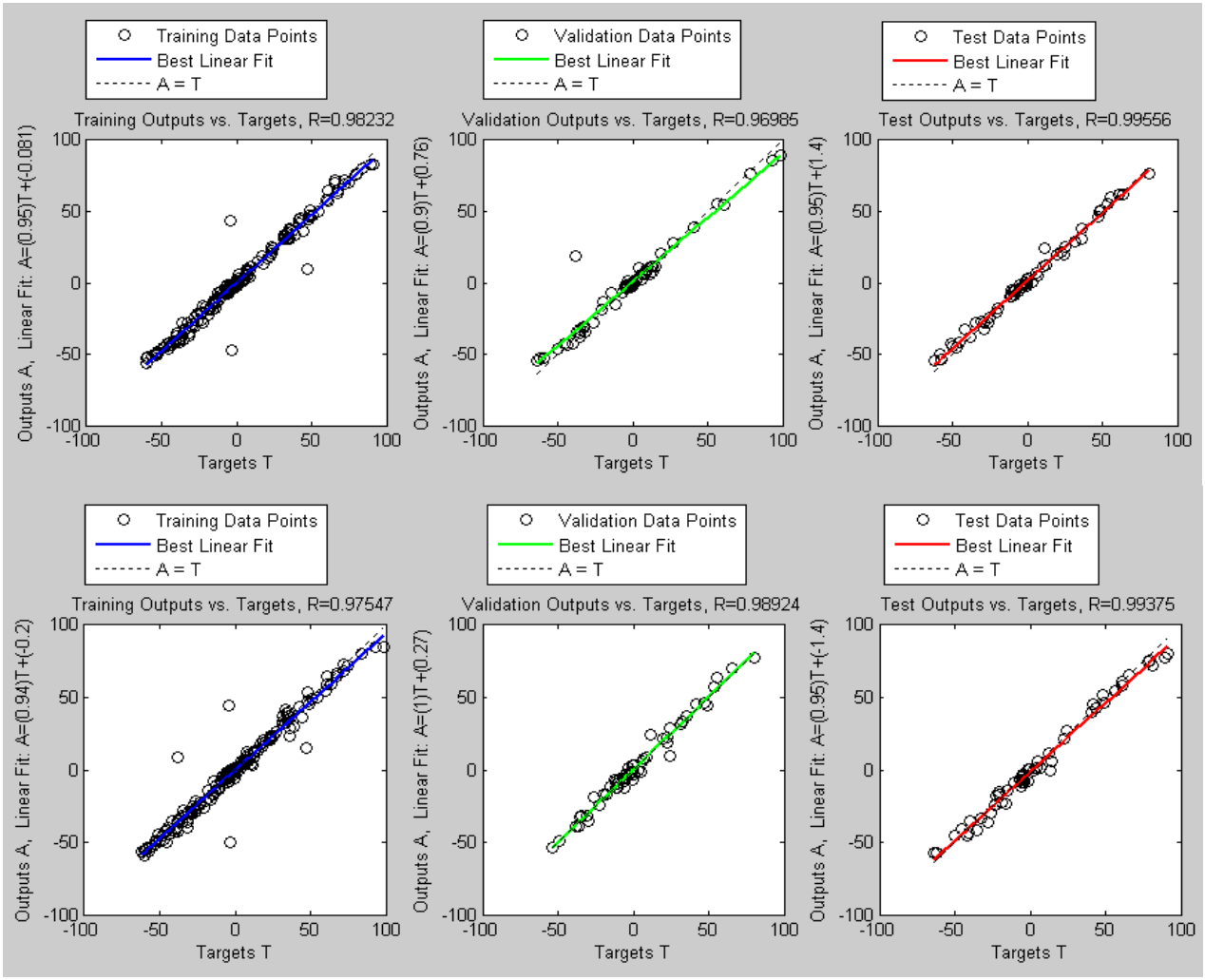

Figure 22 – Dependences of network outputs on experimental data for a network built for output b*; number of neurons in the hidden layer = 5

All calculations were carried out with the ratio of the training and validation sequences set at 20% each. The graphs on the left and at the top correspond to the results obtained exclusively using the nftool, while the graphs on the right and at the bottom correspond to the division of the sequences according to the same first method.

Study results

An analysis of the features of transforming one color model into another based on the construction of ICC profiles was conducted. The fundamentals of profiling display, input, and output devices were outlined. The basic concepts of artificial neural network theory were presented, and the capabilities of data approximation using such networks were considered. In the practical part of the study, numerical modeling of neural networks was performed for approximating data obtained from the measurement of color targets. The influence of network tuning parameters on solving the problem of smoothing experimental data was experimentally investigated.

Conclusions

The possibilities and efficiency of applying artificial neural networks for the approximation of measured data in the construction of color profiles for printing equipment were investigated.

The results of the study were tested during real commissioning works of printing equipment at "Infol" LLC, presented at international scientific conferences, and published [5].

It was practically demonstrated that the implementation of the developed methodology for prepress preparation and equipment adjustment significantly reduces both the cost and time required for performing these tasks.

Thus, the objectives of the study were achieved, and all research tasks were successfully completed.

References.

1. ICC.1:2004-10 Specification (Profile version 4.2.0.0). Color management technology in images – architecture, profile format, and data structure.

2. Hung, P.-C. (1993). Colorimetric calibration in electronic imaging devices using a look-up-table model and interpolations. Journal of Electronic Imaging, 2(1), 53-61.

3. ISO 3664:2000. Standard. Conditions for visual display. Graphic technology and photography.

4. ISO 12646:2004. Standard. Printing technology. Displays for image color verification. Characteristics and conditions for visual display.

5. Azarenkov, V.I., & Deineko, Z.V. (2024). Study of the use of artificial intelligence in the tasks of building digital profiles of printing equipment. Scientific achievements of contemporary society. (p. 271-273).