IMPROVEMENT OF NEURAL NETWORK MODEL FOR PPG SIGNAL ANALYSIS

Zlatko Radovanović[1], **PhD Student**
Alfa BK University, Belgrade, Serbia

Vojkan Nikolić, PhD
University of Criminal Investigation and Police Studies, Belgrade, Serbia

ABSTRACT

Purpose: This paper proposes to improve the method of predicting the age of the subject based on the recorded PPG signal.

Design/Methods/Approach: The PPG signal was obtained by the photoplethysmography method. The prediction method is based on the concept of a neural network, which uses wavelet transformation coefficients of the recorded PPG signal for training. The paper uses an RNN (Recurrent Neural Network) network, which was designed on five different architecture models. The focus of the work is on the comparison of the test results of these five different models and the selection of the model with the most acceptable results for further work. These five RNN models differ in the number of hidden layers. The first model has one hidden layer, the second has two hidden layers, etc., and the fifth model has five hidden layers. A database with 500 recorded PPG signals is used to train the network.

Findings: The results of this paper open the discussion of whether a larger number of hidden layers affects the prediction results and to what extent. There is also a discussion about whether to use RNN architecture or perhaps CNN (Convolutional Neural Network), which is again the topic of some future work in which a comparison of the results of these two architectures should be made.

Originality/Value: The use of wavelet transformation in the analysis of PPG signals can determine in a compromise way when certain changes occur in the signal and what their frequency content is. The goal of this work is in the domain of development, testing, and addition of software solutions in the analysis of PPG signals, application of these solutions in artificial NN modeling and analysis of prediction results obtained by this method.

Keywords: photoplethysmography (PPG), wavelet transformation, neural networks, machine learning, characteristic parameters.

About the authors

Zlatko Radovanović is a doctoral student at Alfa BK University – Faculty of Information Technologies. He graduated in electrical engineering in the field of automation. His areas of interest include signal analysis, system modeling, and development and improvement of NN, and AI.

Vojkan Nikolić, PhD, is an associate professor at the University of Criminal Investigation and Police Studies, Department of Information Technology, Belgrade. His areas of interest include information systems, BI, data mining, and NLP.

1 zlajo@pobox.sk

PURPOSE

This paper will attempt to improve the concept of a neural network, which is used for data prediction based on the recorded PPG signal.

The subject of analysis are signals, i.e., biomedical signals obtained by the method of photoplethysmography – PPG signals (Hertzman, A., 1938).

Biomedical signals, or biosignals, are spatial, temporal, or spatio-temporal records of a biological phenomenon. Biosignals contain information that is of great importance for understanding the specific physiological mechanisms of biological phenomena or the systems from which they originate.

Signals provide important information about the internal state of organs, response to external stimuli, general condition, general health, and the state of various other parameters and are an indispensable part of modern medical diagnostic practice (Allen, J., & Kyriacou, P., 2021).

A photoelectric transducer placed on the skin measures the volume pulse in the peripheral parts of blood vessels (Avolio A., 2002). Corresponding volume pulsations modulate the exposed light beam of incoming infrared light, resulting in changes in intensity. They are measured with a phototransistor/photocell/photoresistor.

Further processing of the intensity changes in the gain stage leads to a visible image of the pulse curve (Allen, J., 2007: 28(3)) (Webster, J. G., 2010) (Allen, J., & Kyriacou, P., 2021). PPG is an easy-to-set-up, efficient device compared to other types of plethysmography (Campolo, 2010).

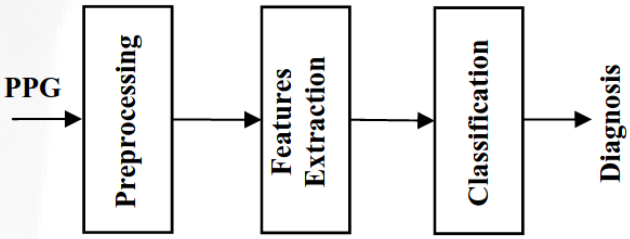

Figure 1: *Processing of PPG signals*

DESIGN/METHODS/APPROACH

PPG signals obtained by the photoplethysmography method are recorded in the database (URL8). The display of the signal obtained by the photoplethysmography method, which was recorded in the database, is shown in Figure 2.

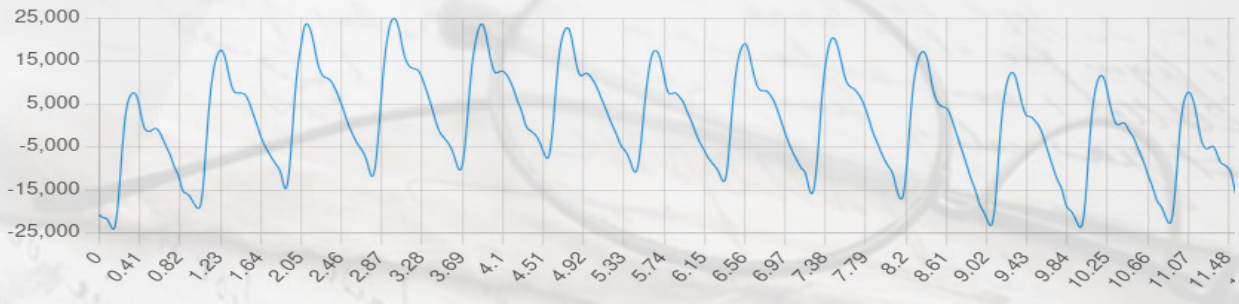

Figure 2: *Display of the signal obtained by the photoplethysmography method*

The signals placed in the database are further analyzed using the wavelet transformation method, and a connection is assumed between the coefficients obtained by the transformation and the parameters important for the diagnosis of the so-called characteristic parameters of the PPG signal (Figure 1) (Radovanović et al., 2025).

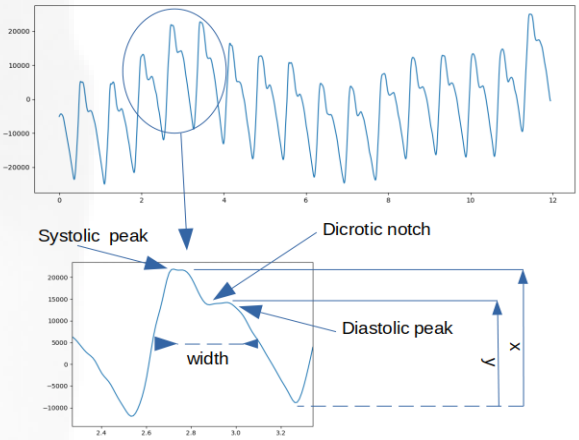

Figure 3: *Characteristic parameters of the PPG signal - systolic, diastolic peak and pulse width*

Characteristic parameters that define and describe the PPG signal and that influenced the choice of wavelet used for transformation (Figure 3) (Radovanović et al., 2025):

Augmentation index – Pressure increase AI is a measure of the impact of the return wave on the systolic arterial pressure, it is obtained by measuring the reflected wave that goes from the periphery to the center.

Pressure increase is obtained using the formula

$$AI = y/x \quad (1)$$

where y is the height of the diastolic and x is the height of the systolic peak during one pulse (Figure 3) (Takazawa, K. et al., 1998).

An alternative formula is also sometimes used in the literature

$$AI = (x-y)/x \quad (2)$$

(Rubins, U. et al., 2008).

The PPG signals (Figure 5) used in this work for training the network are transformed signals using the wavelet transformation method, which was performed using the wavelet rbio3.3 (Figure 4), which was selected as the most optimal for estimating the characteristic parameter AI (Augmentation index) (Radovanović et al., 2025).

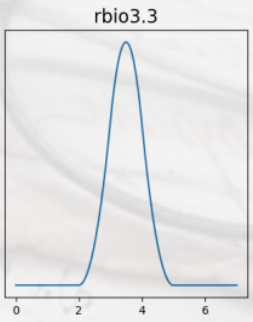

Figure 4: *Wavelet rbio3.3*

Wavelet transformation is often used for the purposes of representing complex signals and extracting their important characteristics (Radunović, D. P., 2005; Mallat, S., 1998; Stark, H. G., 2005) (URL1).

This transformation is an important link in the development of various techniques that can be significant for specific applications, such as clinical practice in medicine (Radovanović et al., 2025).

This procedure requires the execution of several scripts written in the Python programming environment. The scripts will be roughly described in the following text, and the special focus will be on the attempt to improve the neural network that would make predictions based on the analyzed PPG signal, which is also the topic of this work.

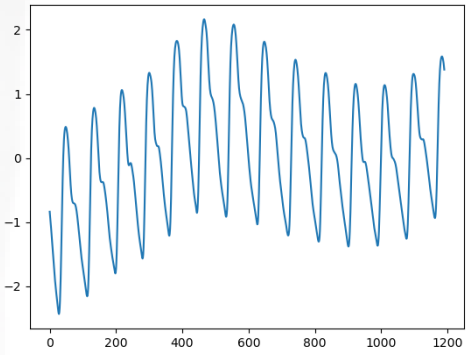

Figure 5: *Typical PPG signal*

Brief description of the procedure for generating the database used for training the neural network (Radovanović et al., 2025):

The recorded PPG signals are located in the database db.xlsx (Figure 2) (URL8).

A script named "excel_to_struct.py" is used to organize the database in which the recorded PPG signals are stored so that they are suitable for further processing, and the result of running this code is a database named "moderated_excel.csv" (Figure 6). Figure 7 shows the drawing of 10 random signals from the "moderated_excel.csv" database.

	A	B	C	D	E	F	G	H	I	J
1	id	age	data							
2	2013338	0.3	[-2075, -1009, 28, 1037, 1990, 2858, 3603, 4160, 4528, 4707, 4745, 4650, 4471, 4245, 400							
3	2013323	0.34	[-11627, -11584, -11670, -12006, -12696, -13626, -14522, -15108, -15323, -15384, -15539,							
4	2018532	0.35	[18174, 17403, 16653, 15922, 15212, 14543, 13914, 13336, 12809, 12312, 11835, 11369,							
5	2018531	0.57	[-13403, -11313, -9301, -7451, -5849, -4402, -2937, -1284, 539, 2235, 3502, 4119, 4188, 38							
6	2018521	0.49	[-18072, -19190, -20238, -21126, -21795, -22065, -21566, -19909, -16744, -12222, -7072, -2							
7	2018515	0.65	[-3533, -4546, -5471, -6206, -6683, -6912, -6961, -6902, -6792, -6653, -6514, -6395, -6335,							
8	2018524	0.58	[-16051, -16536, -17030, -17525, -18051, -18587, -19144, -19721, -20298, -20835, -21288,							
9	2018532	0.35	[18174, 17403, 16653, 15922, 15212, 14543, 13914, 13336, 12809, 12312, 11835, 11369,							
10	2018514	0.65	[-396, -1145, -1885, -2625, -3365, -4068, -4727, -5304, -5801, -6252, -6685, -7145, -7650, -							
11	2018522	0.49	[-8398, -8751, -9104, -9466, -9838, -10209, -10609, -11027, -11473, -11956, -12476, -13043							
12	2018511	0.46	[16104, 15677, 15260, 14865, 14502, 14181, 13925, 13754, 13648, 13583, 13530, 13466,							
13	2018490	0.31	[-25000, -23521, -21599, -18827, -14824, -9936, -4976, -717, 2391, 4427, 5499, 5729, 5313							
14	2018489	0.59	[-15756, -16114, -16442, -16681, -16810, -16870, -16950, -17129, -17437, -17786, -18064,							
15	2018488	0.49	[9998, 9615, 9161, 8555, 7736, 6659, 5386, 3988, 2510, 1041, -356, -1584, -2590, -3374, -							
16	2018487	0.4	[-11070, -9440, -7910, -6580, -5520, -4760, -4250, -3980, -3900, -3960, -4110, -4310, -4520							
17	2018485	0.57	[-8132, -8581, -9040, -9490, -9949, -10417, -10886, -11354, -11833, -12320, -12817, -13324							
18	2018482	0.54	[7120, 6449, 5789, 5166, 4601, 4103, 3643, 3203, 2791, 2379, 1987, 1584, 1192, 828, 483							
19	2018478	0.3	[3244, 3092, 2940, 2747, 2523, 2300, 2117, 2026, 2016, 1955, 1731, 1264, 655, 66, -370, -							

Figure 6: *Signals suitable for further processing are located in the database "moderated_excel.csv"*

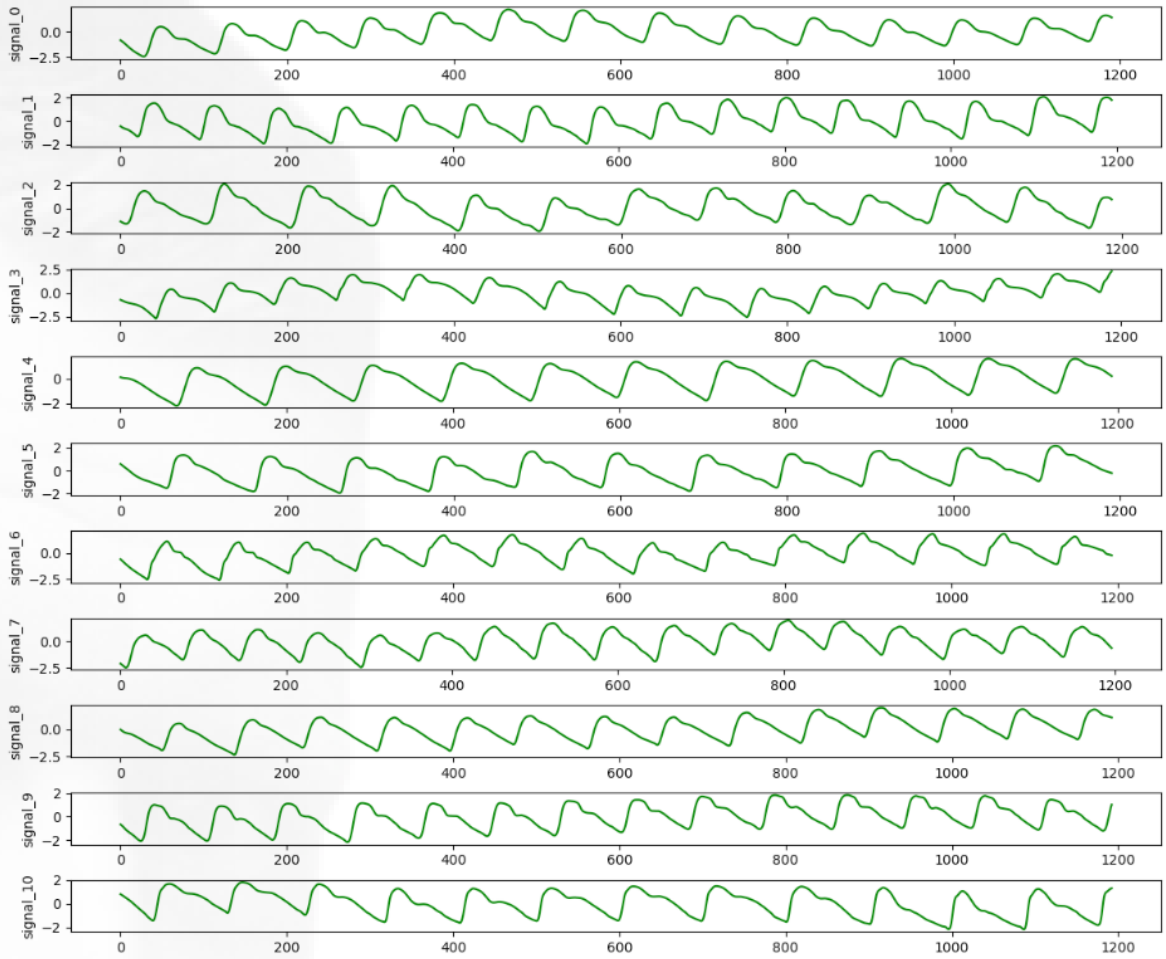

Figure 7: *The drawing of 10 random signals from the "moderated_excel.csv" database*

Brief description of the signal processing and analysis procedure (Radovanović et al., 2025):

The recorded PPG signals (Figure 7), which after sorting the database were placed in the database named "moderated_excel.csv" (Figure 6), were transformed using the wavelet transformation with the help of the code named "wavedec_csv.py". This code generates a database named "wavelet_transformed_rbio3.3_level_7_db1_500.csv".

The database "wavelet_transformed_rbio3.3_level_7_db1_500.csv" contains the transformed PPG signals used for training the network (Radovanović et al., 2025).

The transformation was performed in the following way:
The wavelet transform of the signal is performed using the Python built-in function wavedec, the inverse transform is performed using the Python function waverec.

The PPG signal was transformed by wavelet transformation, the transformation coefficients "0, 1, 4, 5, 6, 7" were modified i.e., overwritten with 0 (Figure 8), then the inverse transformation of the modified signals is performed, so that the reconstructed signal is used for training the network.

For the wavelet transformation in the Python programming environment, a library called PyWavelets was used. This library provides a comprehensive set of functions and tools for wavelet analysis and wavelet-based signal processing and also enables various wavelet transforms of signals, including discrete (dwt, wavedec), continuous (cwt), and inverse transforms.

PyWavelets supports a wide range of predefined groups of wavelets, such as Daubechies, Haar, Coiflets, Symlets, rbior, and others. Each of these groups of wavelets has different characteristics and provides flexibility in choosing the appropriate wavelet function for analysis. PyWavelets provides tools for plotting wavelet transforms of signals and wavelet coefficients. Wavelet functions and spectrograms can also be plotted (URL2) (URL4) (URL5).

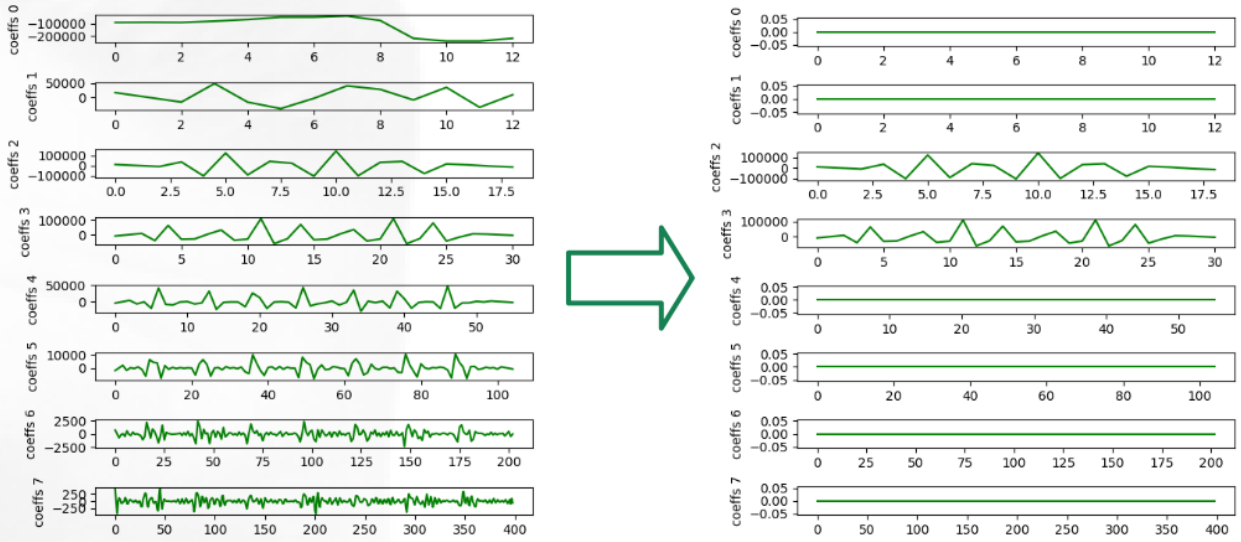

Figure 8: *Modification of wavelet transform coefficients*

A neural network consists of sets of artificial neurons (often called layers) that process data and perform complex tasks such as classification, regression, shape recognition, etc. In Python, neural networks (URL6) can be implemented using various libraries such as TensorFlow, Keras, PyTorch, and others (URL2).

General procedure for implementing a neural network in the Python programming environment (URL2) (URL6) (URL7):
Installation of libraries needed to work with neural networks.

Preparation and loading of data that will be used for neural network training. The data should be in a format suitable for the neural network, for example, strings from the NumPy library or DataFrame from the pandas library.

Defining the network architecture that corresponds to the specific task. This includes the choice of layers, the number of neurons in each layer, the choice of activation functions, etc. Network implementation can be done using library functions or by defining a network class intended for a specific task.

Initialization of network parameters such as weight and bias. Different initialization methods can be used, such as random initialization or pre-trained weights.

Network training can be performed using a forward and back propagation algorithm (e.g., gradient descent). It is also necessary to define the loss function that would be minimized during training. The network needs to be trained iteratively, adjusting the weights and bias towards optimal values.

Validation and testing. After the training, with the help of the data set for validation, the network's performance is evaluated. Evaluation metrics such as accuracy, precision, responsiveness, or other metrics can be used. Also, the network should be tested on an independent data set to evaluate its overall performance.

Optimization and setting of parameters. If the network does not achieve certain performance, the procedure can be repeated with different network parameters, such as the number of layers (topic of this work), number of neurons, learning speed, regulation, dropout, etc. Adjusting these parameters can improve network performance.

A brief description of the libraries and functions used in the Python programming environment for generating and testing neural networks (URL4) (URL5) (URL6):

pandas: Pandas is a library for data manipulation and analysis. It provides data structures such as DataFrame (a tabular data format) and Series (a one-dimensional data structure). In this paper, pandas is used to load data from a CSV file, manipulate the data, and convert the data into a suitable format for training a neural network.

numpy: NumPy is a library for numerical operations in Python. It provides efficient data structures for working with multidimensional arrays and matrices, as well as a large number of mathematical functions for data manipulation. In this paper, NumPy is used to convert data to NumPy arrays and perform numerical operations during training and evaluation of a neural network.

sklearn.model_selection.train_test_split: train_test_split is a function from scikit-learn that is used to split data into training and testing sets. This function randomly splits the data in a certain ratio between the training and testing sets. In this paper, train_test_split is used to split the data into training and testing input data sets to evaluate the performance of the model.

tensorflow.keras.models.Sequential: Sequential is a class from Keras, which is part of the TensorFlow deep learning library. This class is used to create sequential neural network models, where layers are added one after the other. In this paper, Sequential is used to define the model architecture.

tensorflow.keras.layers.Dense: Dense is a Keras class that represents a layer with fully connected neurons in a neural network. A layer with fully connected neurons means that all neurons in the previous layer are connected to every neuron in the current layer. In this paper, Dense is used to add fully connected layers to the model.

tensorflow.keras.preprocessing.sequence.pad_sequences: pad_sequences is a function from the Keras library used to add padding to sequences. This function ensures that all sequences have the same length by padding shorter sequences with zeros. In this paper, pad_sequences is used to add padding to transformed data sequences.

FINDINGS

In this paper, the neural network concept is based on the RNN (Recurrent Neural Network) ((URL3) Goodfellow et al., 2016: 367–415) network, which is designed on five different architecture models. Each model represents a network with one, two, three, four, and five layers. The number of layers to be tested is the subject of optimization and parameter setting. Each layer has a different number of neurons (Table 1).

For validation, metrics from the tensorflow library will be used, which are suitable for regression networks and prediction of continuous values (URL9):

MAE (Mean Absolut Error): Calculates the average of the absolute differences between the true values (y_i) and the predicted values ($\hat{y}_i$).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (3)$$

MSE (Mean Squared Error): Calculates the average of the squared differences between the true and predicted values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (4)$$

The results of the prediction of the neural network represent a type of validation of the method based on the wavelet transform.

Tabular representation of the architecture of the five neural network models that will be tested (Table 1):

Table 1: *Configuration view of five different tested models*

Model	Input layer	First hidden layer	Second hidden layer	Third hidden layer	Fourth hidden layer	Fifth hidden layer	Output layer
1	100	30					1
2	64	32	16				1
3	64	32	16	8			1
4	64	32	16	8	4		1
5	128	64	32	16	8	4	1

GENERATING RNN MODELS

The following code shows the generation of an RNN training model. It could be said that training a prediction network and testing it on the same data is actually a methodological error: a model that just repeated the patterns it was just familiar with would have a perfect score, but it would fail to predict anything useful on the data it was unfamiliar with. This situation is "jargonically" called "overfitting". To avoid this, it is common practice when performing (supervised) network training to keep part of the available data as a test data set. The following code shows the training of the network, where the transformed PPG signal file is used for training:

input file (transformed PPG signals):

csv_waverec_file = wavelet_transformed_rbior3.3_level_7_db1_500.csv'

output generated models for five configurations:

model with one hidden layer:

output_file_1 = '../model/model_1_epoch_100_batch_size_8.h5';

model with two hidden layer:

output_file_2 = '../model/model_2_epoch_100_batch_size_8.h5';

model with three hidden layer:

output_file_3 = '../model/model_3_epoch_100_batch_size_8.h5';

model with four hidden layer:

output_file_4 = '../model/model_4_epoch_100_batch_size_8.h5';

model with five hidden layer:

output_file_5 = '../model/model_5_epoch_100_batch_size_8.h5';

Below is the code in the python programming environment where the models are defined:

```
import ast
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import pad_sequences
from keras.callbacks import History
history_1 = History()
history_2 = History()
history_3 = History()
history_4 = History()
history_5 = History()
# load the .csv file
csv_waverec_file = '../csv_wavelet_transformed/wavelet_transformed_rbior3.3_level_7_db_1_500_1.csv';
df = pd.read_csv(csv_waverec_file);
# Convert 'Coefficients' from string to list using ast.literal_eval
df['data'] = df['data'].apply(lambda x: ast.literal_eval(x) );
sequences = df['data'].values;
# padding
X = pad_sequences( sequences,
        maxlen=2000,
        padding='post' );
y = df['age'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,random_state=42);
# Model architecture remains the same
model_1 = Sequential()
model_2 = Sequential()
model_3 = Sequential()
model_4 = Sequential()
model_5 = Sequential()

#model_1
# Add the input layer
model_1.add(Dense(100, input_dim=len(X[0]), activation='tanh'))
# Add the first hidden layer
model_1.add(Dense(30, activation='sigmoid'))
# Add the output layer
model_1.add(Dense(1, activation='linear'))
#model_2
# Add the input layer
model_2.add(Dense(64, input_dim=len(X[0]), activation='tanh'))
# Add the first hidden layer
model_2.add(Dense(32, activation='sigmoid'))

# Add the second hidden layer
model_2.add(Dense(16, activation='sigmoid'))
```

```
# Add the output layer
model_2.add(Dense(1, activation='linear'))
#model_3
# Add the input layer
model_3.add(Dense(64, input_dim=len(X[0]), activation='tanh'))
# Add the first hidden layer
model_3.add(Dense(32, activation='sigmoid'))
# Add the second hidden layer
model_3.add(Dense(16, activation='sigmoid'))
# Add the third hidden layer
model_3.add(Dense(8, activation='sigmoid'))
# Add the output layer
model_3.add(Dense(1, activation='linear'))
#model_4
# Add the input layer
model_4.add(Dense(64, input_dim=len(X[0]), activation='tanh'))
# Add the first hidden layer
model_4.add(Dense(32, activation='sigmoid'))
# Add the second hidden layer
model_4.add(Dense(16, activation='sigmoid'))
# Add the third hidden layer
model_4.add(Dense(8, activation='sigmoid'))
# Add the fourth hidden layer
model_4.add(Dense(4, activation='sigmoid'))
# Add the output layer
model_4.add(Dense(1, activation='linear'))
#model_5
# Add the input layer
model_5.add(Dense(128, input_dim=len(X[0]), activation='tanh'))
# Add the first hidden layer
model_5.add(Dense(64, activation='sigmoid'))
# Add the second hidden layer
model_5.add(Dense(32, activation='sigmoid'))
# Add the third hidden layer
model_5.add(Dense(16, activation='sigmoid'))
# Add the fourth hidden layer
model_5.add(Dense(8, activation='sigmoid'))
# Add the fifth hidden layer
model_5.add(Dense(4, activation='sigmoid'))
# Add the output layer
model_5.add(Dense(1, activation='linear'))
# Model compile
model_1.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])
model_2.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])
model_3.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])
model_4.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])
model_5.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])
```

```
history_1 = model_1.fit(X_train, y_train, epochs=100, batch_size=8,
        validation_data=(X_test, y_test), callbacks=[history_1] )
loss_1 = model_1.evaluate(X_test, y_test)
print('Test loss:', loss_1)

history_2 = model_2.fit(X_train, y_train, epochs=100, batch_size=8,
        validation_data=(X_test, y_test), callbacks=[history_2] )
loss_2 = model_2.evaluate(X_test, y_test)
print('Test loss:', loss_2)

history_3 = model_3.fit(X_train, y_train, epochs=100, batch_size=8,
        validation_data=(X_test, y_test), callbacks=[history_3] )
loss_3 = model_3.evaluate(X_test, y_test)
print('Test loss:', loss_3)

history_4 = model_4.fit(X_train, y_train, epochs=100, batch_size=8,
        validation_data=(X_test, y_test), callbacks=[history_4] )
loss_4 = model_4.evaluate(X_test, y_test)
print('Test loss:', loss_4)

history_5 = model_5.fit(X_train, y_train, epochs=100, batch_size=8,
        validation_data=(X_test, y_test), callbacks=[history_5] )
loss_5 = model_5.evaluate(X_test, y_test)
print('Test loss:', loss_5)
# Save the model
output_file_1 = '../model/model_1_epoch_100_batch_size_8.h5';
output_file_2 = '../model/model_2_epoch_100_batch_size_8.h5';
output_file_3 = '../model/model_3_epoch_100_batch_size_8.h5';
output_file_4 = '../model/model_4_epoch_100_batch_size_8.h5';
output_file_5 = '../model/model_5_epoch_100_batch_size_8.h5';
model_1.save(output_file_1);
model_2.save(output_file_2);
model_3.save(output_file_3);
model_4.save(output_file_4);
model_5.save(output_file_5);
print ('model_1 saved to', output_file_1);
print ('model_2 saved to', output_file_2);
print ('model_3 saved to', output_file_3);
print ('model_4 saved to', output_file_4);
print ('model_5 saved to', output_file_5);
plt.close('all')
plt.figure()
plt.plot(history_1.history['loss'])
plt.plot(history_2.history['loss'])
plt.plot(history_3.history['loss'])
plt.plot(history_4.history['loss'])
plt.plot(history_5.history['loss'])
plt.title('model loss')
```

```
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['one hidden layer', 'two hidden layer', 'three hidden layer',
    'four hidden layer', 'fife hidden layer'], loc='upper right')
plt.show()
plt.figure()
plt.plot(history_1.history['mae'])
plt.plot(history_2.history['mae'])
plt.plot(history_3.history['mae'])
plt.plot(history_4.history['mae'])
plt.plot(history_5.history['mae'])
plt.title('model mae')
plt.ylabel('mae')
plt.xlabel('epoch')
plt.legend(['one hidden layer', 'two hidden layer', 'three hidden layer',
    'four hidden layer', 'fife hidden layer'], loc='upper right')
plt.show()
```

Algorithm 1: *The code in the python programming environment where the models are defined*

The result of executing this code is generated prediction models in *.h5 format. Graphs were also generated that show the ratio of loss and MAE values depending on the number of epochs (Figure 9) (Figure 10).

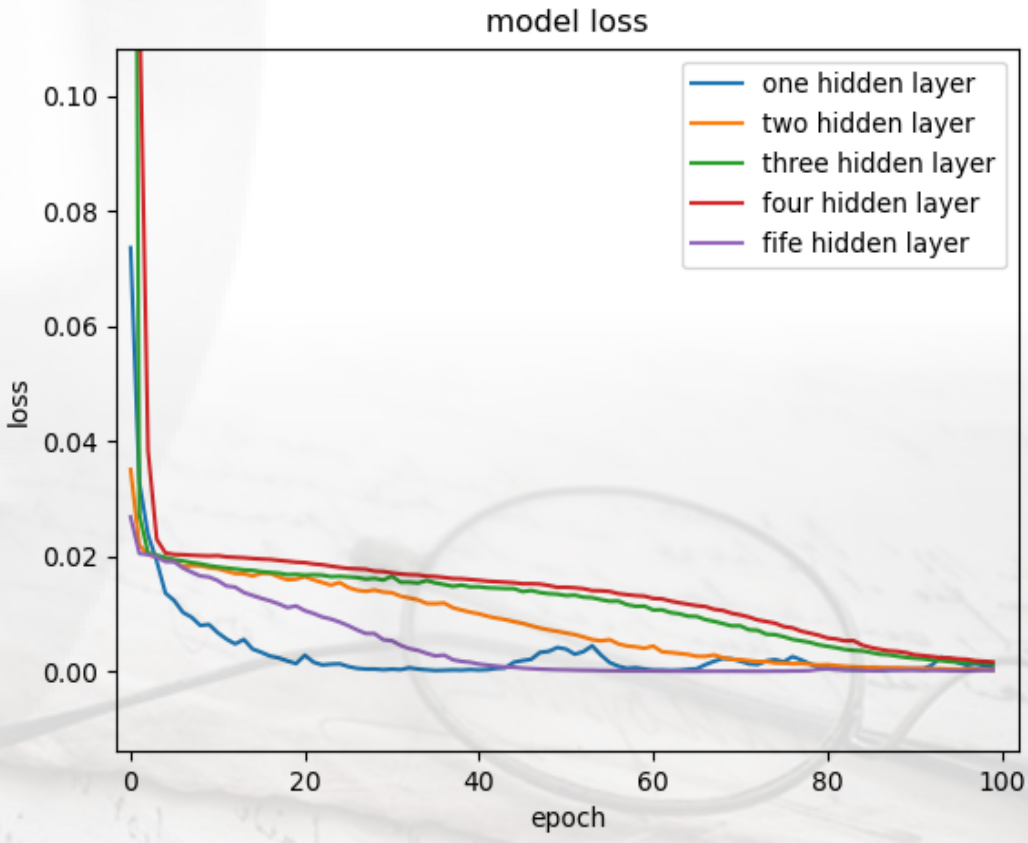

Figure 9: *Loss values for five different models*

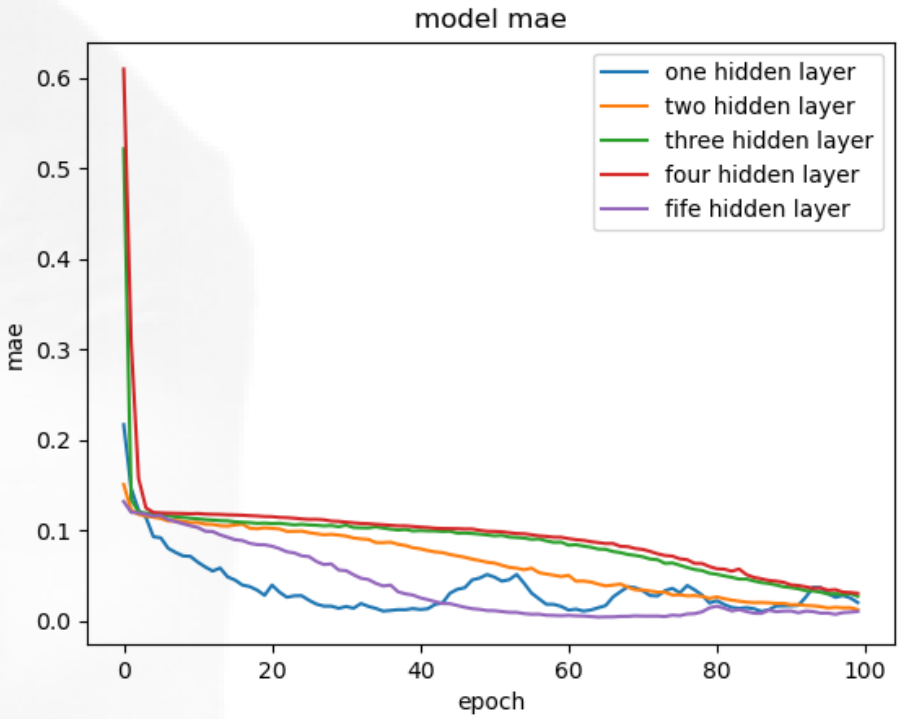

Figure 10: *MAE values for five different models*

TESTING RNN MODELS

Python code for predicting the subject's age based on the PPG signal transformed by wavelet transformation. Testing the generated model from the previous section will be performed on the PPG signal file:

input file (transformed PPG signals):

csv_waverec_file = 'wavelet_transformed_rbior3.3_level_7_db1_500.csv'

output files:

model with one hidden layer:

outputfile_11 = '../prediction/prediction_model_1_epoch_100_batch_size_8.csv'

model with two hidden layer:

outputfile_21 = '../prediction/prediction_model_2_epoch_100_batch_size_8.csv'

model with three hidden layer:

outputfile_31 = '../prediction/prediction_model_3_epoch_100_batch_size_8.csv'

model with four hidden layer:

outputfile_41 = '../prediction/prediction_model_4_epoch_100_batch_size_8.csv'

model with five hidden layer:

outputfile_51 = '../prediction/prediction_model_5_epoch_100_batch_size_8.csv'

Below is the code in the python programming environment for testing five different prediction models:

```
import csv
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.utils import pad_sequences
from tensorflow.keras.models import load_model
from sklearn.metrics import mean_absolute_error, mean_squared_error

# load the model
model_1 = load_model('../model/model_1_epoch_100_batch_size_8.h5');
model_2 = load_model('../model/model_2_epoch_100_batch_size_8.h5');
model_3 = load_model('../model/model_3_epoch_100_batch_size_8.h5');
model_4 = load_model('../model/model_4_epoch_100_batch_size_8.h5');
model_5 = load_model('../model/model_5_epoch_100_batch_size_8.h5');

# load the .csv file
csv_waverec_file = '../csv_wavelet_transformed/wavelet_transformed_rbior3.3_level_7_db_1_500_1.csv';
input_data = pd.read_csv(csv_waverec_file);

# extract data from the DataFrame
ppg_signals_wt = input_data['data'].apply(eval).tolist();
correct_ages = input_data['age'].values*100;
data_id = input_data['id'].values;
ppg_signals_wt = pad_sequences(ppg_signals_wt, maxlen = 2000, padding = 'post');

# predict the age for each PPG signal
ages_predicted_1 = model_1.predict(ppg_signals_wt)*100;
ages_predicted_2 = model_2.predict(ppg_signals_wt)*100;
ages_predicted_3 = model_3.predict(ppg_signals_wt)*100;
ages_predicted_4 = model_4.predict(ppg_signals_wt)*100;
ages_predicted_5 = model_5.predict(ppg_signals_wt)*100;
outputfile_11 = '../prediction/prediction_model_1_epoch_100_batch_size_8.csv'
outputfile_21 = '../prediction/prediction_model_2_epoch_100_batch_size_8.csv'
outputfile_31 = '../prediction/prediction_model_3_epoch_100_batch_size_8.csv'
outputfile_41 = '../prediction/prediction_model_4_epoch_100_batch_size_8.csv'
outputfile_51 = '../prediction/prediction_model_5_epoch_100_batch_size_8.csv'

def write_prediction (correct_ages, ages_predicted, outputfile, model):
 y_true = []; y_pred = [];
 with open(outputfile, 'w', newline ='') as output_file:
   mae = mean_absolute_error(correct_ages, ages_predicted)
   mse = mean_squared_error(correct_ages, ages_predicted)

   writer = csv.writer(output_file);
   writer.writerow(['data_id', 'actual_age', 'predicted_age']); # write header row

   for i in range(len(correct_ages)):
     print(f"data_id:{(data_id[i])}, Actual Age: {round(correct_ages[i])},
       Predicted Age: {round(ages_predicted[i][0])}");
```

```
# write the data to the output csv file
    writer.writerow( [data_id[i], round(correct_ages[i]),
        round(ages_predicted[i][0])] );
    y_true.append(round(correct_ages[i]) );
    y_pred.append(round(ages_predicted[i][0]) );

  writer.writerow(['Mean Absolute Error:', round(mae)]);
  writer.writerow(['Mean Squared Error:', round(mse)]);

 print(f"Mean Absolute Error: {round(mae)}")
 print(f"Mean Squared Error: {round(mse)}")
 print ('prediction saved to', output_file);

 fig, ax = plt.subplots(figsize=(10,5))
 plt.title(f"{model}")
 ax.plot(range(len(y_true)), y_true, '-b',label='Actual')
 ax.plot(range(len(y_pred)), y_pred, '-r', label='Predicted')
 ax.legend(loc="lower right")
 plt.show()
 return mae, mse, y_true, y_pred
plt.close('all')
mae_11, mse_11, y_true_1_11, y_pred_1_11 = write_prediction(correct_ages, ages_predicted_1,
        outputfile_11, 'model 1 \n one hidden level')
mae_21, mse_21, y_true_1_21, y_pred_1_21 = write_prediction(correct_ages, ages_predicted_2,
                outputfile_21, 'model 2 \n two hidden level')
mae_31, mse_31, y_true_1_31, y_pred_1_31 = write_prediction(correct_ages, ages_predicted_3,
                outputfile_31, 'model 3 \n three hidden level')
mae_41, mse_41, y_true_1_41, y_pred_1_41 = write_prediction(correct_ages, ages_predicted_4,
                outputfile_41, 'model 4 \n four hidden level')
mae_51, mse_51, y_true_1_51, y_pred_1_51 = write_prediction(correct_ages, ages_predicted_5,
                outputfile_51, 'model 5 \n five hidden level')
mae = [mae_11, mae_21, mae_31, mae_41, mae_51];
mse = [mse_11, mse_21, mse_31, mse_41, mse_51];
hidden_layer = [1, 2, 3, 4, 5]
fig, ax = plt.subplots(figsize=(10,5))
ax.plot(hidden_layer, mae, '-b',label='MAE')
ax.plot(hidden_layer, mse, '-r', label='MSE')
ax.legend(loc="upper right")
plt.show()
```

Algorithm 2: *The code in the python programming environment for testing five different prediction models*

As a result of executing this code, a table of MAE and MSE values is generated for each defined model (Table 2). A graphic representation of the dependence of the number of hidden layers and the values of MAE and MSE is shown in Figure 16. Five graphs are also generated where the current value and prediction are shown for each defined model (Figure 11) (Figure 12) (Figure 13) (Figure 14) (Figure 15).

Table 2: *MAE and MSE testing results for five different models*

MODEL	MAE	MSE
1	3	36
2	4	44
3	4	41
4	5	54
5	3	33

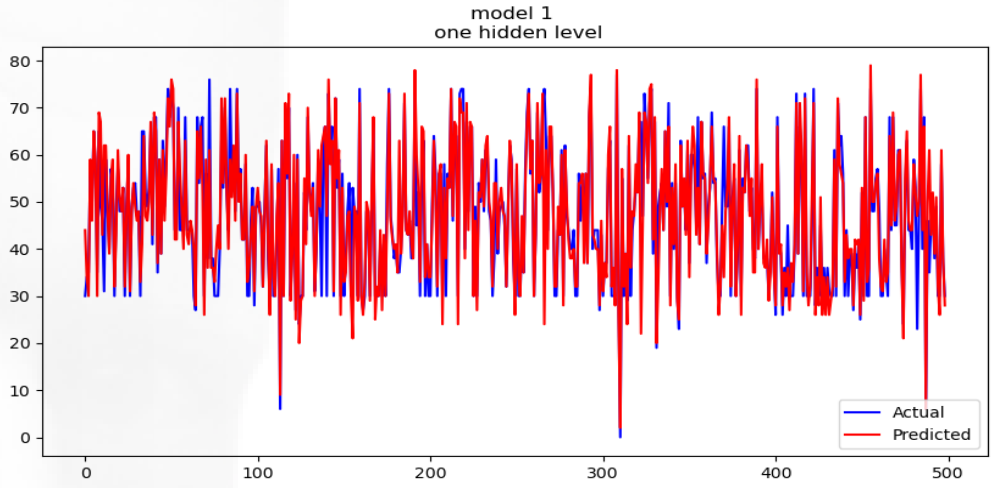

Figure 11: *Display of actual values and predictions for a model with one hidden layer*

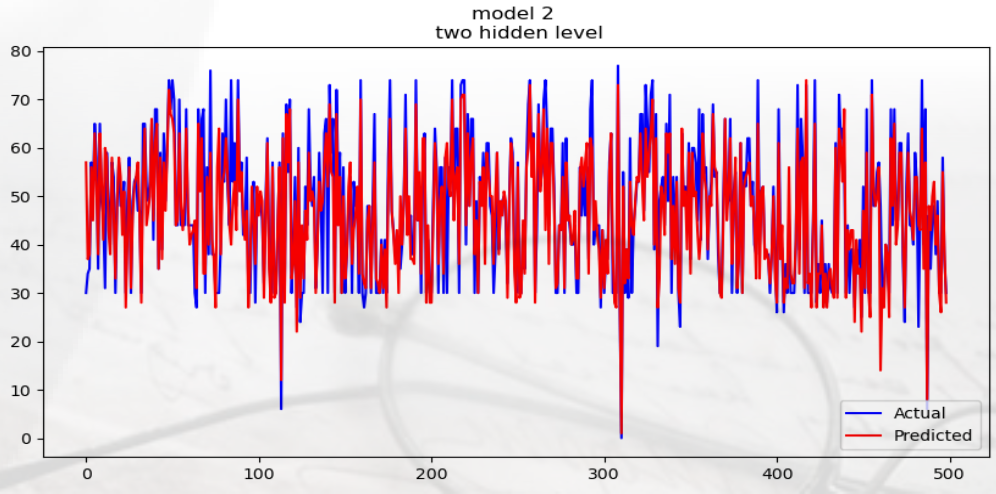

Figure 12: *Display of actual values and predictions for a model with two hidden layers*

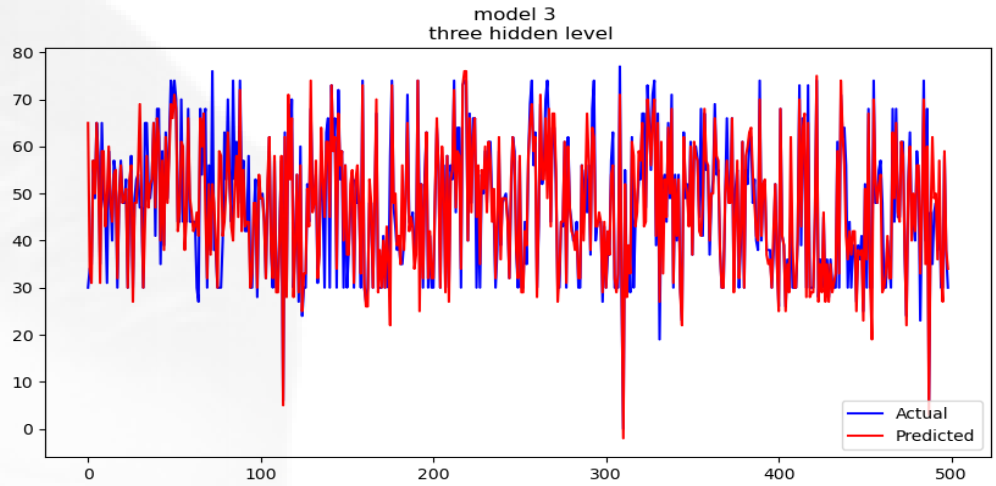

Figure 13: *Display of actual values and predictions for a model with three hidden layers*

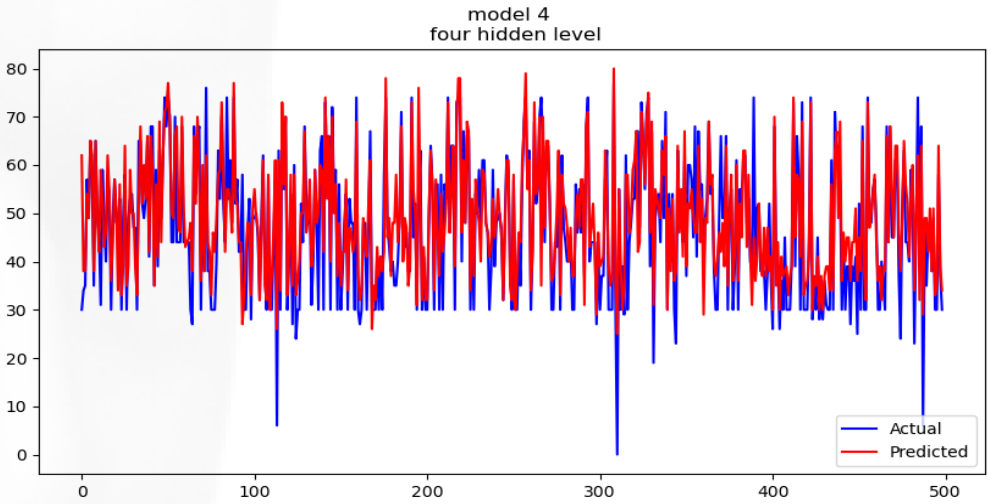

Figure 14: *Display of actual values and predictions for a model with four hidden layers*

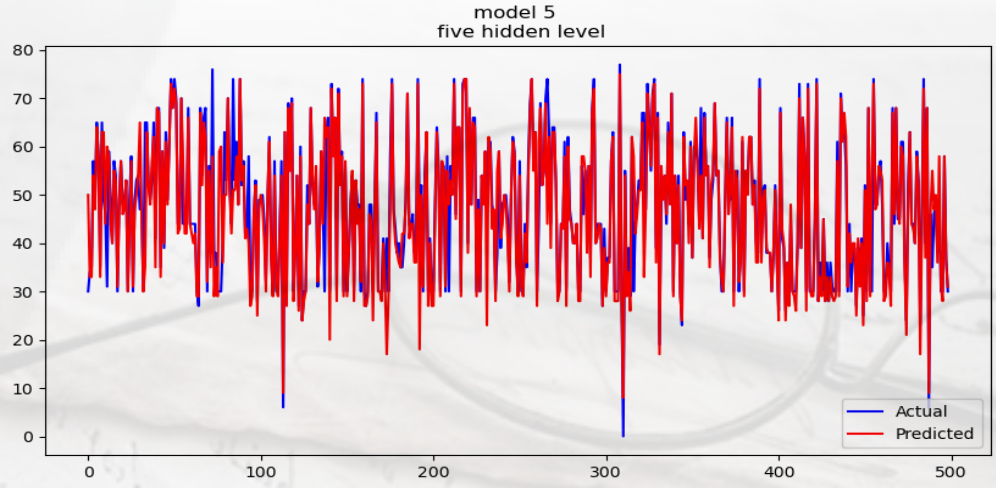

Figure 15: *Display of actual values and predictions for a model with five hidden layers*

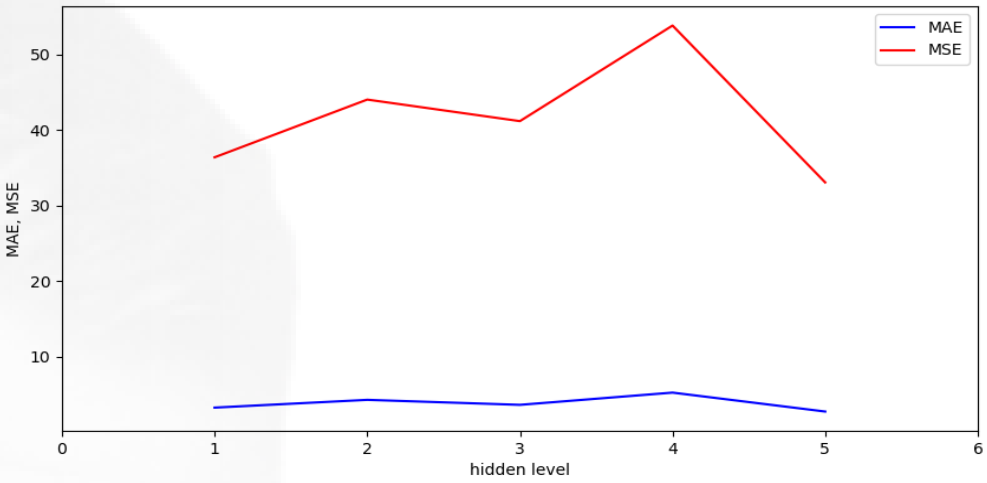

Figure 16: *Display of MAE and MSE values in relation to the number of hidden layers*

ORIGINALITY/VALUE

This paper presents the concept of an RNN network for prediction based on the recorded PPG signal. The PPG signal was transformed by wavelet transform. The network concept is set on five models, where each model has one layer more than the previous one.

In this way, the performance of the network was tested depending on the number of hidden layers. It is important to note that for regression prediction of continuous PPG signal values, it is necessary to adequately prepare the data, properly define the network architecture, train the network on a large data set, and regularly monitor and adjust the model parameters in order to achieve the best possible performance (URL2).

From the testing performed in this paper and based on the generated graphs and tabular displays, it can be concluded that the model with five hidden layers is more accurate than the other models. The topic of further work is testing with a larger data set.

If the graphic analyzed in figures 9, 10, and 16 is considered, a conclusion can be reached that the model with a larger number of hidden layers is more accurate at a smaller number of epochs and that the precision equalizes as the number of epochs increases. If the graphic is analyzed in figure 11, 12, 13, 14, and 15, it can be concluded that the actual value and the prediction have the most overlap in models with a larger number of hidden layers.

A discussion is opened for further work and testing on whether to increase the number of epochs or the number of hidden layers, training the network on a diverse and representative data set, which is very important for analyzing PPG signals, testing and setting up the CNN (Convolutional Neural Network) network for analyzing and classifying PPG signals, and comparing CNN results with RNN results.

REFERENCES

Radovanović, Z., Jokić, S., Jokić, I., Gerazov, B., Kovačević, A. & Gligorić, N. (2025). PPG signal analysis and wavelet selection for feature extraction. *ALFATECH – Smart cities and modern technologies*. Proceedings (p. 220). Belgrade: Alfa BK University. DOI: 10.46793/ALFATECHproc25.220R

Allen, J. (2007). Photoplethysmography and its application in clinical physiological measurement. *Physiological Measurement*, April 2007, 28(3). DOI:10.1088/0967-3334/28/3/R01

Nikolic, V., Markoski, B., Kuk, K., Randjelovic, D., Cisar, P. (2017) . Modelling the System of Receiving Quick Answers for e-Government Services: Study for the Crime Domain in the Republic of Serbia. *ACTA POLYTECHNICA HUNGARICA*, 2017, vol. 14, No. 8, pp. 143–163 DOI:10.12700/APH.14.8.2017.8.8

Popovic, S., Viduka, D., Basic, A., Dimic, V., Djukic, D., Nikolic, V., Stokic, A. (2025). Optimization of Artificial Intelligence Algorithm Selection: PIPRECIA-S Model and Multi-Criteria Analysis. *ELECTRONICS*, (2025), vol. 14, No. 3. DOI:10.3390/electronics14030562

Maksimovic, A., Nikolic, V., Vidojevic, D., Randjelovic, M., Djukanovic, S., Randjelovic, D. (2024). Using Triple Modular Redundancy for Threshold Determination in DDOS Intrusion Detection Systems. *IEEE ACCESS*, (2024), vol. 12. DOI:10.1109/ACCESS.2024.3384380

Stolic, P., Stevic, Z., Petronic, S., Nikolic, V., Stevic, M., Kreculj, D., Milosevic, D. (2024). Modeling, Simulation, and Computer Control of a High-Frequency Wood Drying System. *ELECTRONICS*, (2023), vol. 12, No. 1. DOI:10.3390/electronics12010226

Kevkic, T., Nikolic, V., Stojanovic, V., Milosavljevic, D., Jovanovic, S., (2022). Modeling electrostatic potential in FDSOI MOSFETS: An approach based on homotopy perturbations. *OPEN PHYSICS*, (2022), vol. 20, No. 1, pp. 106–116. DOI:10.1515/phys-2022-0012

Raičević, S., Nikolić, V. (2024). Comparative analysis of certain clustering algorithms that do not require a predefined number of clusters on the articles of the criminal code of the Republic of Serbia. *Journal of Computer and Forensic Sciences*. 2024, vol. 3, No. 2, pp. 28–42.

Milenković, J., Pavlović, M., Nikolić, V., Jasić, A., Premceski, V. (2020). Example of Clustering Using K-Means Method in Python. *International Conference on Information Technology and Development of Education – ITRO 2020*, Technical Faculty "Mihajlo Pupin" Zrenjanin.

Janićijević, S., Nikolić, V. (2021). Graph Structures for Data Visualizations. *Serbian Journal of Engineering Management*. 2021, Vol. 6, No. 2. DOI: 10.5937/SJEM2102024J

Avolio, A. (2002). The finger volume pulse and assessment of arterial properties. *Journal of Hypertension,* 2002-12, Vol. 20 (12), 2341-2343. DOI: 10.1097/00004872-200212000-00007

Webster, J., G. (2010). *Medical Instrumentation: application and design*. Wiley 2010.

Campolo, D. (2017). *New Developments in Biomedical Engineering*. InTech 2017.

Hertzman, A. (1938). The blood supply of various skin areas as estimated by the photoelectric plethysmograph. *American journal of physiology*. 124 (2), 328-40.

Takazawa, K., T., N., Fujita, M., Matsuoka, O., Saiki, T., Aikawa, M., Tamura, S. & Ibukiyama C., (1998). Assessment of vasocative agents and vascular aging by the second derivative of photoplethysmogram waveform. *Journal of Hypertension*. 32, 65–70. DOI: 10.1161/01.hyp.32.2.365

Rubins, U., Grabovskis, A., Grube, J. & Kukulis, I. (2008). *Photoplethysmography Analysis of Artery Properties in Patients with Cardiovascular Diseases.* Springer Berlin Heidelberg.

Allen, J., & Kyriacou, P. (2021). Photoplethysmography – Technology, Signal Analysis and Applications. eBook ISBN: 9780128235256.

Mallat, S. (1998). *A wavelet Tour of Signal Processing*. Academic Press, New York.

Radunović, D., P. (2005). *Talasići*. Akademska misao, Belgrade.

Stark, H., G. (2005). *Wavelets and Signal Processing*. University of Applied Sciences, Germany. Springer Berlin Heidelberg.

INTERNET SOURCES

(URL1) Misiti, M., Misiti, Y., Oppenheim, G. & Poggi, J., M. (1997). Wavelet Toolbox for Use with MATLAB, MathWorks. https://de.mathworks.com/help/wavelet/

(URL2) http://ataspinar.com/posts/machine-learning-with-signal-processing-techniques/

(URL3) Goodfellow, I., Bengio, Y. & Courvill, A. (2016). Deep Learning. e-book. MIT Press. https://www.deeplearningbook.org/contents/ml.html

(URL4) https://github.com/PyWavelets/

(URL5) https://pywavelets.readthedocs.io/

(URL6) https://scikit-learn.org/

(URL7) https://github.com/stevanjokic

(URL8) https://play.google.com/store/apps/details id=srb.ctb.pulse.heartrate.camera.ecg4everybody (the application through which the PPG signals used in this work were collected and generated in a file)

(URL9) https://apxml.com/courses/getting-started-with-tensorflow/chapter-4-training-evaluating-models/compiling-metrics