ORNL/TM-2024/3295

Assessment of Machine Learning for Ultrasonic Nondestructive Evaluation of Alkali-Silica Reaction in Concrete

Hongbin Sun
Samantha Sabatino

March 2024

Draft. Document has not been reviewed and approved for public release.

OAK RIDGE National Laboratory

ORNL IS MANAGED BY UT-BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2024/3295

Light Water Reactor Sustainability Program

M2LW-24OR0403023

Nuclear Energy and Fuel Cycle Division

ASSESSMENT OF MACHINE LEARNING FOR ULTRASONIC NONDESTRUCTIVE EVALUATION OF ALKALI–SILICA REACTION IN CONCRETE

Hongbin Sun
Samantha Sabatino

March 2024

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831
managed by
UT-BATTELLE LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AI	artificial intelligence
ASR	alkali-silica reaction
ASR-2D	alkali-silica reaction sample with 2D confinement
DL	deep learning
DNN	deep neural network
NDE	nondestructive evaluation
ML	machine learning
PCA	principal component analysis
RMSE	root mean square error
SVM	support vector machine
SVR	support vector regression

ABSTRACT

Alkali–silica reaction (ASR) is a type of material degradation in concrete structures that leads to concrete cracking and rebar corrosion, thereby reducing the material's structural integrity and the overall structure's lifetime and raising safety concerns. Ultrasonic nondestructive evaluation (NDE) has been proven to be a valuable technique for assessing concrete properties and monitoring ASR progression in concrete. However, the deployment and analysis of ultrasonic NDE and its data requires specialized expertise, often relying on the engineer's subjective interpretation. With the surge in computational power, artificial intelligence (AI) and machine learning (ML) algorithms have become popular in automating NDE data analysis. Various industrial sectors are increasingly adopting ML algorithms for NDE data analysis with a growing emphasis on AI–assisted automation. Regulatory agencies are also preparing for this technological shift, anticipating corresponding revisions in standards. Thus, there is an urgent need to identify the capabilities and limitations of current ML technologies for the evaluation of concrete material properties and damage status. Furthermore, the effects of various factors on ML model performance must be thoroughly investigated.

The study summarized herein evaluated the effectiveness of two ML models (i.e., support vector regression (SVR) and deep neural network (DNN)) in predicting concrete material damage induced by ASR based on the long-term ultrasonic monitoring data. Four distinct concrete specimens were cast with artificially induced ASR, and over a period exceeding 500 days, ultrasonic signals and expansion data were continuously collected. For the SVR model, wave velocity and 12 other wave features were extracted from the ultrasonic signals, with 6 out of 13 features selected as input for the model. Different combinations of training and testing datasets were designed to explore factors influencing prediction performance, including the range of data within training and testing sets, in addition to various signal preprocessing methodologies. These findings suggest the importance of using a training dataset with a broader data range compared with testing datasets for improved model performance alongside consistent signal preprocessing across datasets.

Additionally, this work studied the effect of temperature on model performance, revealing significant prediction errors when the temperature of testing data differed from that of training data. Furthermore, this study examined training and testing datasets from two distinct specimen batches, revealing the SVR model's superior generalization ability compared with DNN when considering data from different batches. Lastly, DNN models were trained and tested using both time-domain signals and frequency spectra. The results showed potential for achieving high regression performance (e.g., the ML model was able to predict ASR expansion with relatively high accuracy), although further efforts in model tuning and data preprocessing are still required to improve performance. Finally, a generic workflow was summarized for the application of ML on automated data analysis for ultrasonic NDE.

Overall, the results and conclusions of this study could provide insights into the capabilities and effectiveness of ML when applied to ultrasonic NDE data and help identify best practices for using ML for the ultrasonic NDE of concrete material properties.

1. INTRODUCTION

1.1 MOTIVATION

Alkali–silica reaction (ASR) is a common form of material degradation found in concrete infrastructure. This reaction occurs when certain types of silica, such as chert or microcrystalline quartz, which are present in the aggregate, interact with alkali metals in the cement. As a result of this interaction, hydrated sodic or K-bearing Ca silicate gels, known as *ASR gels*, are formed [1]. These gels absorb water and expand within the material's pores and preexisting cracks, creating internal pressure and causing the material to expand. Over time, this expansion leads to the development and spread of additional cracks within the concrete. Consequently, ASR contributes to the deterioration of concrete, leading to a loss of mechanical properties and increased penetration of harmful substances. Ultimately, this process can lead to the corrosion of steel reinforcement and negatively affect the service life of the structure.

In 2009, ASR was discovered in certain below-grade structures at the Seabrook Nuclear Power Plant, and later, ASR was confirmed to exist in multiple concrete structural buildings, including the electrical tunnel, containment enclosure building, residual heat removal vault, emergency diesel generator building, and emergency feedwater building. The safety issues and concerns caused by ASR-induced concrete damages hinder the license renewal of nuclear power plants. To ensure the continued safe operation of nuclear power plant concrete structures over extended periods, it is imperative to conduct research that comprehensively examines the long-term effect of ASR on the durability, serviceability, and safety of these structures [2]. This research will provide essential technical insights for evaluating license renewals of nuclear power plants.

Currently, visual inspection is the major condition assessment method for concrete structures with ASR. However, visual inspection is subjective and does not yield accurate information about internal concrete damage. Concrete coring with subsequent petrographic examination is another common method for assessing the presence of ASR [3], but the petrographic examination process is time-consuming, labor-intensive, and destructive. To overcome these drawbacks, nondestructive evaluation (NDE) methods, such as ultrasonic testing, offer a promising solution for evaluating ASR damage in concrete. Various ultrasonic wave parameters, including wave velocity, attenuation, and amplitude, have been used to quantify ASR damage [4], [5], [6]. As ASR progresses, the amplitude and velocity of ultrasonic waves propagating through the concrete decrease while the wave attenuation increases. Multiple studies have implemented ultrasonic NDE for long-term monitoring of the ASR damage development in concrete and have proven the success of ultrasonic testing techniques for assessing a concrete structure that has developed ASR damage [2], [7], [8], [9], [10].

The analysis of ultrasonic NDE data demands specialized expertise in ultrasonic testing and material properties, relying heavily on engineering experience and knowledge. However, recent advancements in artificial intelligence (AI) and machine learning (ML) have revolutionized this process, offering automated solutions for material characterization and damage detection. Industrial sectors are increasingly embracing ML algorithms to streamline NDE data analysis, with a growing emphasis on AI-assisted automation. Regulatory agencies are also gearing up for this shift, anticipating revisions in corresponding standards. Recent research has extensively explored ML applications in NDE, particularly in ultrasonic testing, with a main focus on damage detection in metal materials.

However, limited literature exists on concrete damage evaluation, a gap that will be explored further in the following literature review part of this report. The growing variety of ML techniques highlights the importance of evaluating the current capabilities in ML-supported analysis of NDE data, especially in the ultrasonic NDE of concrete. One primary objective of this work is to identify the capabilities and limitations of current ML technologies and investigate the effects of various factors on the ML model

performance. The results of this study could provide insights into the capabilities and effectiveness of ML when applied to ultrasonic NDE data and help to identify best practices for using ML on the ultrasonic NDE of concrete.

1.2 LITERATURE REVIEW

As computational power increases, ML has been widely applied to the automated analysis of ultrasonic data to characterize the condition of concrete materials. Previous works have mainly focused on using ML algorithms to predict concrete strength based on single wave parameters, such as ultrasonic pulse velocity and signal amplitude [11], [12], [13], [14], along with measurements from other nondestructive methods (e.g., rebound hammer, penetration resistance, visual inspection). However, the approaches used in the previous literature only use a single wave parameter, such as the ultrasonic pulse velocity, while neglecting other valuable information embedded in both time-domain and frequency-domain ultrasonic signals. In recent years, several studies have been conducted using ML and NDE to evaluate the ASR damage development in concrete. Clayton et al. [15] proposed using ML algorithms to process the ultrasonic linear array data collected on concrete specimens to evaluate the ASR damage development. Ai et al. [16] implemented deep learning (DL) models, convolutional neural networks, and autoencoders to evaluate the ASR damage and classify the two phases during the development based on acoustic emission NDE data. Miele et al. [17], [18] used ML and vibroacoustic testing to localize the ASR damage in concrete specimens. Shin et al. [19] successfully identified the initiation date of the ASR development in concrete using autoencoders and ultrasonic monitoring signals. In previous works [20], [21], the authors of this report have studied multiple ML models (e.g., linear regression, support vector machine (SVM), neural networks) for ASR expansion prediction using long-term ultrasonic monitoring signals from two concrete specimens.

A limited number of works have focused on the use of ML for ultrasonic NDE of concrete, particularly in assessing ASR damage. However, during the literature review process, a diverse array of studies was uncovered regarding ML applications in ultrasonic NDE for detecting welding defects. The insights obtained from these studies can also be of significant benefit to the application of ML for the NDE of concrete. Valuable lessons have been gathered from examining ultrasonic NDE approaches aimed at the automatic classification of welding defects. In the literature exploring SVM models for classifying welding defects, wave parameters, and statistical features were extracted from both time-domain and frequency-domain signals to serve as input for the SVM models [22], [23], [24], [25], [26], [27]. This same methodology is applicable to shallow neural network models, which also use these extracted features as input [28], [29], [30], [31], [32]. In contrast, deep neural network (DNN) models can directly use either time-domain signals or frequency spectra as model input [33], [34], [35], [36]. However, in this scenario, the model's input layer may contain an excessive number of input nodes, requiring precautions to prevent overfitting. Strategies such as batch normalization and the use of dropout layers should be considered in such cases.

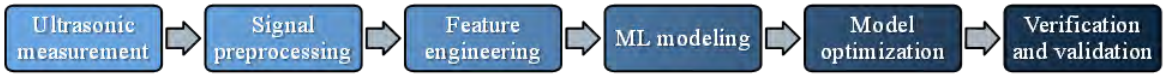

Figure 1-1. Summary of the workflow of using ML for ultrasonic NDE.

The literature review may suggest a standardized workflow for employing ML in ultrasonic NDE, as illustrated in

Figure 1-1. This workflow is adaptable for various applications within ultrasonic NDE, such as assessing ASR damage in concrete or classifying welding defects. The following list outlines key insights gleaned from the literature review to aid in the application of ML for the NDE of ASR damage in concrete.

1. Limited studies have specifically addressed the use of ML for evaluating ASR damage through NDE techniques. Nonetheless, numerous research endeavors have used ML in NDE or ultrasonic NDE applications. Insights from these studies can be directly leveraged in the context of employing ML for evaluating ASR damage in concrete.

2. When employing shallow ML models, input features typically include wave parameters and statistical parameters derived from time-domain signals and frequency spectra. Extracting these features requires a comprehensive understanding of raw signals, and careful feature selection may be necessary.

3. In the case of deep ML models, both time-domain and frequency spectra can serve as direct model inputs. However, additional effort is required for model tuning to optimize performance effectively.

4. Various preprocessing techniques are essential to clean the input data, with the specific procedure contingent upon the nature of the input. Tailoring preprocessing procedures to suit the input type is crucial for enhancing model accuracy.

5. Employing multiple performance metrics is advisable because a single metric may not accurately portray model performance. Using a range of metrics provides a comprehensive assessment of the model's effectiveness.

1.3 OBJECTIVES OF THIS WORK

This report aims to evaluate the effectiveness of two ML models—namely, support vector regression (SVR) and DNN—in predicting ASR expansion based on extensive, long-term monitoring of ultrasonic data across multiple specimens. Initially, four distinct concrete specimens were cast with artificially boosted ASR. Over a period exceeding 500 days, ultrasonic signals and expansion data were continuously collected. For the SVR model, wave velocity, along with various other wave features, were extracted from the ultrasonic signals. With feature selection techniques, features demonstrating a significant correlation with expansion were identified and used as input for the model.

Compared to the previous report [21], this work focused more on the factors that may impact the model performance for expansion prediction based on the ultrasonic data and dived deep into the DNN models with full waveform as the model input. Different combinations of training and testing datasets were designed to explore factors influencing prediction performance. These factors include the range of data within training and testing sets, as well as various signal preprocessing methodologies. Additionally, the effect of temperature on model performance was studied, given the sensitivity of ultrasonic signals to environmental and specimen temperatures. Furthermore, the report details the examination of training and testing datasets sourced from two distinct batches of specimens. This investigation aimed to evaluate the model's generalization ability, assessing its efficacy in extrapolating from one batch to another. Lastly, attention was directed toward exploring DNN models trained using both time-domain signals and frequency spectra, aiming to assess the capability of DL models for ultrasonic NDE and the factors that determine model performance.

2. BACKGROUND: MACHINE LEARNING

This section provides an introduction to commonly used ML terminology.

Artificial intelligence (AI) is a specialized field within computer science dedicated to designing systems and machinery with the capacity to execute activities typically associated with human intelligence [37]. These activities include learning, problem-solving, decision-making, language comprehension, and perception. ML, a subset of AI, focuses on constructing algorithms and statistical models to empower computers to learn from data and generate predictions or decisions. Unlike traditional programming approaches, *machine learning* (ML) systems refine their performance in specific tasks by discerning patterns and insights from data. *Deep learning* (DL) represents a subset of ML characterized by the use of multilayered neural networks to derive knowledge from extensive datasets.

Although a variety of ML applications are possible, most common applications may be categorized into problems associated with regression, classification, or segmentation. This work focuses on regression using ML algorithms. *Regression* refers to the use of ML to learn relationships between input and output with the goal of predicting continuous valued output variables from a given set of inputs. In NDE, regression-based models may be used, for instance, in estimating the concrete strength from a set of NDE measurements, such as ultrasonic velocity. *Classification* refers to categorizing input data into predefined classes or categories. The goal of classification is to train a model that can accurately assign new instances of data to one of these classes based on the features or attributes provided, such as “Intact” or “Damaged” for NDE of concrete. *Segmentation*, usually in the context of image analysis, refers to the use of ML or statistical methods to label individual image pixels into one of two classes.

Model training refers to the process of instructing an ML algorithm or model to make predictions or decisions by exposing it to a dataset known as the *training dataset*, which contains various examples. In supervised learning, this dataset consists of input data, which can either be raw or processed measurements, along with corresponding output labels or target values. Alternatively, unsupervised learning does not have labeled data; instead, it uses input data without any label information, with the training process aimed at grouping or clustering the data based on inherent patterns or similarities. Reinforcement learning, the third learning paradigm, focuses on learning optimal actions to maximize a cumulative reward. This approach is often employed in optimization tasks, such as optimal design and data-driven control. In this work, only supervised learning was used.

The training process entails iteratively adjusting the model’s parameters or weights to minimize the disparity between its predictions and the actual target values present in the training data. Before initiating training, an ML model is constructed, and both its architecture (model structure) and hyperparameters are defined. *Hyperparameters* in ML are parameters set before training that are not learned from the training data. An example of a hyperparameter is the *learning rate*, which determines the pace at which the model learns from data. Setting the learning rate too low can lead to a sluggish convergence of the model to optimal weights, minimizing the difference between predictions and actual target values. Conversely, a learning rate that is too high may cause the model to oscillate around the optimal set of weights. Thus, hyperparameters, like the learning rate, significantly influence the control of the learning process. The adjustment of the model structure and hyperparameters to attain optimal model performance is known as *model tuning* or *hyperparameter tuning*.

During the training process, the *validation* dataset serves as an independent dataset to assess the model’s performance. The model’s performance on the validation dataset helps to tune hyperparameters, avoid overfitting, and determine if the model is generalizing well to new, unseen data. *Overfitting* is a phenomenon in which the model learns the training data too well to the point that it begins to capture

noise or random fluctuations in the data. As a result, the overfitted model performs exceptionally well on the training data but fails to generalize effectively to new, unseen data.

Model training necessitates the specification of a *loss function*, which quantifies the disparity between the model's prediction (output) and the desired output value. Commonly employed loss functions in ML applications encompass root mean square error (RMSE), mean absolute error, and binary cross-entropy loss. The selection of a specific loss function is contingent upon the problem type (classification versus regression) and its characteristics. Each loss function imposes varying penalties on the model when its prediction deviates from the desired output value. During model training, multiple epochs are typically employed, during which the model's parameters gradually adapt to minimize training error and enhance performance. An epoch denotes a single pass through the entire training dataset. Upon completing an epoch, the model has processed and learned from all training data. Depending on the learning algorithm used, parameter adjustments may occur after the model has processed all training data (a batch), a subset of training data (a mini-batch), or a single example from the training data. Upon completing training, the model is evaluated using a *testing dataset*. Model testing, or inference, involves assessing the generalization performance to ascertain if the model can accurately predict or decide on new, unseen data that were not encountered during training.

This work focused on the application of ML (SVR and DNN) for regression based on ultrasonic signals. The report presents the results of an empirical assessment of ML data analysis for the prediction of ASR expansion using ultrasonic long-term monitoring data, focusing on the factors influencing ML regression performance. The results of these assessments are expected to be useful to industries and the US Department of Energy for future activities involving the NDE of concrete.

3. CONCRETE SPECIMENS AND ULTRASONIC DATA COLLECTION

Four concrete specimens (i.e., 2 of each *ASR* and *ASR-2D* [alkali–silica reaction sample with 2D confinement]) were cast with artificial ASR development. Two of them were large specimens with the same 0.3 × 0.3 × 1.12 m dimensions, as shown in Figure 3-1 (ASR and ASR-2D). The other two were small specimens with dimensions of 0.3 × 0.3 × 0.61 m (small ASR and small ASR-2D in the figure). All four specimens used the same mix design, which contained a reactive coarse aggregate. Along with the reactive coarse aggregate, NaOH was added to boost the alkali content to 1.50% equivalent Na_2O by mass of cement, which promotes rapid ASR in the concrete specimens; the mix design is shown in Table 3-1. Whereas no reinforcing steel bars were added to the ASR specimen to allow free expansion, headed steel rebar was installed in the ASR-2D specimen in the longitudinal (y) and vertical (z) directions to confine the expansion in these directions. More details about the concrete specimens can be found in the study by Malone [2], [38]. After 28 days, the specimens were moved to a conditioning chamber, where they were kept at 38°C and 90% humidity for more than 500 days to accelerate the ASR development. Demountable mechanical strain gauge targets were installed on each surface, and the expansions in the three directions were measured using demountable mechanical strain gauges (Mayes Instruments Limited, United Kingdom) every two weeks when the chamber was temporarily shut down. The volumetric expansion is the summation of the expansions in the three directions. Figure 3-2 shows the volumetric expansion histories of the ASR, ASR-2D, small ASR, and small ASR-2D specimens for the experiment duration (i.e., 552 days). Numerous surface cracks developed on four specimens, which indicate the success of the ASR promotion in the specimens. The two small specimens only have an expansion record of 330 days and showed very consistent expansion over the total curing period.

Table 3-1. Mix design of the ASR specimens

Component	Quantity
Cement (kg/m^3)	350
Water (kg/m^3)	175
Coarse aggregate (kg/m^3)	1,039
Fine aggregate (kg/m^3)	839
Water reducer (mL/kg)	2.3
50%/50% NaOH (kg/m^3)	9.31

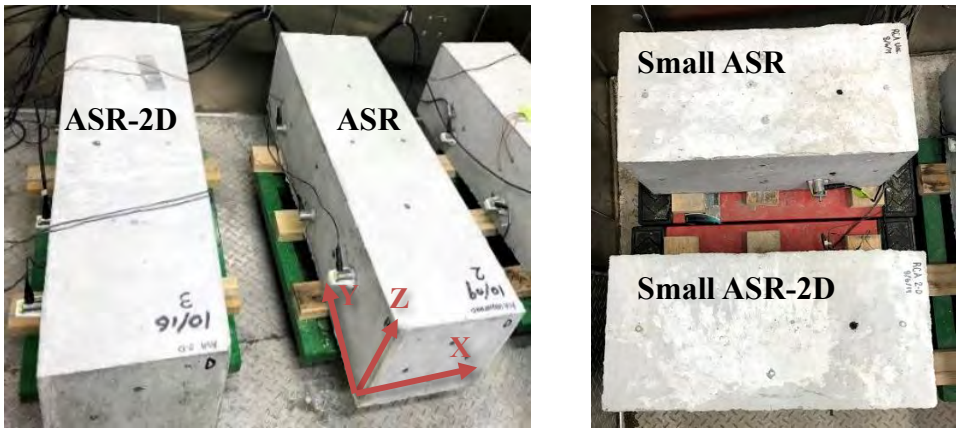

Figure 3-1. (left) ASR, ASR-2D concrete specimens and (right) small ASR, small ASR-2D specimens.

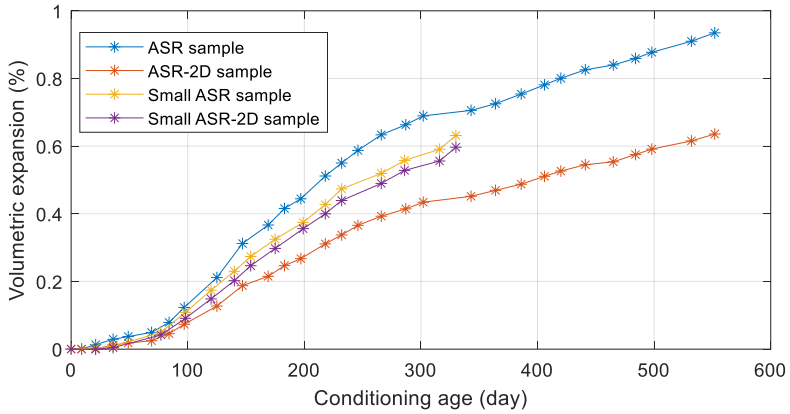

Figure 3-2. Volumetric expansions for the ASR, ASR-2D, small ASR, and small ASR-2D specimens.

A pair of ultrasonic transducers was installed on the concrete specimen permanently for long-term monitoring (see Figure 3-3). A 15 mm diameter piezoceramic disc (STEMINC Inc.) was used as the transmitter, and an acoustic emission sensor (R15I-AST, Physical Acoustic, New Jersey) with a center frequency of 150 kHz was used as the ultrasonic receiver. A multiplexer was used to switch between different transmitter and receiver pairs. The received signals were digitized by an oscilloscope (PICO4262) with a sampling rate of 10 MHz and averaged 100 times to increase the signal-to-noise ratio. The ultrasonic signal was collected every 12 hours, along with temperature measurements. Data were not collected during the conditioning chamber shutdowns. More details about the ultrasonic monitoring system can be found in other works by Sun [7], [39]. Overall, 644 ultrasonic signals were collected from the ASR specimen, and 620 signals were collected from the ASR-2D specimen. Because more ultrasonic signals were recorded than expansion measurements (29 data points), the expansion corresponding to each ultrasonic signal was calculated using linear interpolation. This linear interpolation is justified because the conditioning environment was consistent between two expansion measurements.

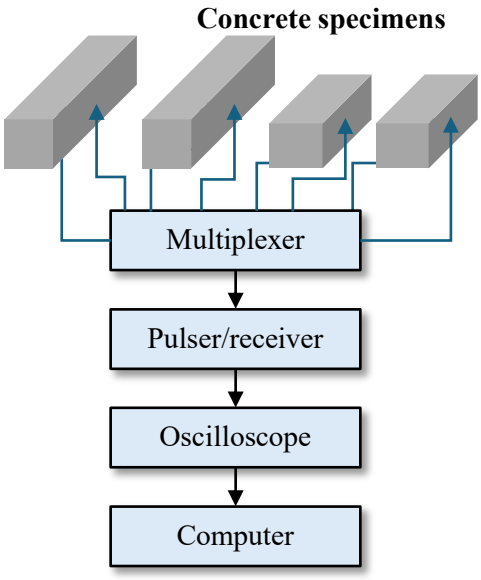

Figure 3-3. Diagram of the ultrasonic monitoring system for the four concrete specimens.

4. ULTRASONIC DATA PROCESSING

Figure 4-1 shows the time-domain signals at concrete ages of 60, 260, and 460 days for the ASR specimen. The first signal [Figure 4-1(a)] was collected at an early age when the concrete specimen exhibited a relatively small expansion; at this time, the ASR may not have been initiated, and the signal still contained high-frequency components. The second signal [Figure 4-1(b)] was collected at 260 days, and the concrete specimen exhibited relatively large expansion because of the ASR. The signal had a much lower signal amplitude and a different pattern compared with the first signal because more energy was attenuated by the concrete cracks resulting from the ASR-induced expansion. The third signal [Figure 4(c)] was collected at the late stage of ASR development with large concrete expansion. The signal also showed a low signal amplitude. These time-domain signals indicate that the ultrasonic signal contains useful information that is related to the status of the ASR development, which is measured by the volumetric expansion. Before extracting wave parameters and statistical features from these time-domain signals, all signals were normalized by scaling the maximum amplitude to one.

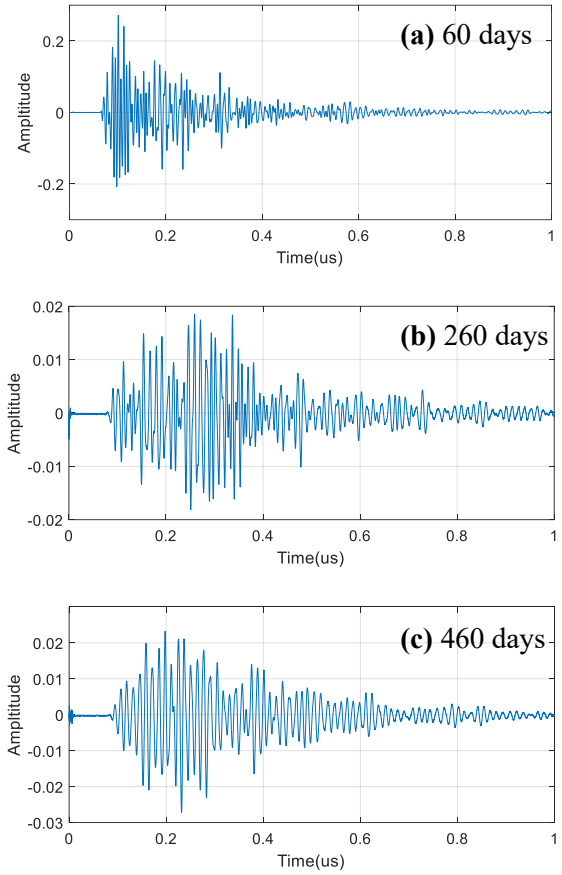

Figure 4-1. Time domain signals at different concrete ages: (a) 60 days, (b) 260 days, and (c) 460 days.

4.1 ULTRASONIC WAVE FEATURES

As discussed in previous work [21], 13 features were extracted from the time-domain signals for ML model training and testing. These features, along with the expansion data from the specimens, were used for model training and testing. Two types of features were extracted from the time-domain signal. The first was wave velocity, which is directly related to the crack density and concrete modulus. The second parameter extracted includes the wavelet features calculated via discrete wavelet transform, which

decomposes the time-domain signal into signals in different frequency bands. After the decomposition, wavelet features are calculated for each band signal.

Wave velocity is the most commonly used parameter in ultrasonic nondestructive testing for material characterization. When the ASR specimen exhibits gradual cracking, the wave propagation velocity is reduced. In this work, wave velocity was extracted as one of the features used for damage assessment. The Akaike information criterion picker was used to detect the arrival time automatically. The Akaike information criterion picker method was originally developed for analyzing the P-wave velocity of seismic waves [40].

Aside from the wave velocity, other features were extracted from the ultrasonic signals in both the time and frequency domains using wavelet transform. Discrete wavelet transform is an algorithm that can be used to quickly obtain the wavelet transform coefficients of a discrete time-domain signal. The algorithm decomposes the ultrasonic signal with a low-pass and high-pass filter into its coarse approximation coefficient cA_1 and detail coefficient cD_1. The bandwidth is $[0, f/2]$ for the approximation coefficient cA_1 and $[f/2, f]$ for the detail coefficient cD_1, and f is half of the sampling rate. This process is repeated until the desired decomposition level is achieved. The wavelet feature extraction is based on these detail coefficients. Each of the ultrasonic time-domain signals was decomposed into seven levels with the MATLAB function *modwt* and "db4" wavelet. Because the energy of the original ultrasonic signal was in the frequency range of 40 to 250 kHz, only the last three levels of the detail coefficients cD_5, cD_6, and cD_7 were used for the feature extraction from discrete wavelets transform. The coefficient cD_5 was in the frequency range of 156–312 kHz, cD_6 was in the range of 78–156 kHz, and cD_7 was in the range of 39–78 kHz. For each of the three detail coefficients, four features were extracted:

(1) Mean amplitude

$$A_m = \frac{1}{n}\sum_{i=1}^{n} |x_i|, \tag{1}$$

where x_i represents the amplitude of each sample in the signal, and n is the total number of samples. As the ASR damage accumulates, the energy decreases in the high-frequency bandwidth and increases in the low-frequency bandwidth. Therefore, the mean value of the detail coefficient will have the same trend as the energy in different bandwidths.

(2) Maximum amplitude

$$A_{\max} = \max(x_i). \tag{2}$$

Similar to the mean amplitude, the maximum amplitude also has the same trend as the energy in different bandwidths.

(3) Total energy

$$E = \sum x_i^2. \tag{3}$$

The energy ratio in the low-frequency bandwidth increases as the ASR develops.

(4) Attenuation coefficient

$$A(t) = A_0 \mathrm{e}^{-\alpha t}, \tag{4}$$

where $A(t)$ is the fitted function for the signal after the maximum amplitude using an exponential decay function. The coefficient α in the equation represents the attenuation. Four feature parameters were extracted from each of the three detail coefficients cD_5, cD_6, and cD_7, leading to 12 wavelet features. The wave velocity history is plotted in Figure 4-2 for the ASR specimen over the curing period. The wave velocity demonstrated a decreasing trend with some variations that represent the sensor reinstallation or abnormal chamber temperature. Therefore, the wave velocity history could be considered noisy. The other 12 wavelet features are plotted in Figure 4-3 for the same specimen. Each subfigure of Figure 4-3 includes three features from the three levels of detailed coefficients. Using the feature selection method discussed in previous work, the wave velocity and five other wavelet features were selected for the model input to achieve better prediction performance. These five wavelet features are in the following list and plotted in bold lines in Figure 4-3.

1. Mean amplitude for cD_6 (78–156 kHz)
2. Maximum amplitude for cD_6 (78–156 kHz)
3. Maximum amplitude for cD_7 (39–78 kHz)
4. Total energy for cD_7 (39–78 kHz)
5. Attenuation for cD_5 (156–312 kHz)

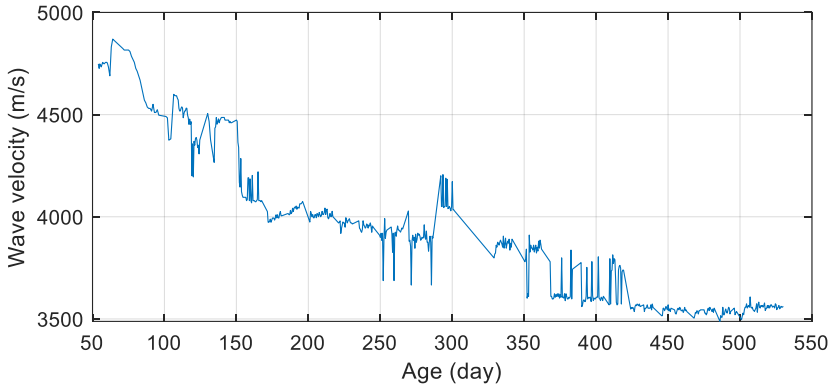

Figure 4-2. Wave velocity history for the ASR specimen.

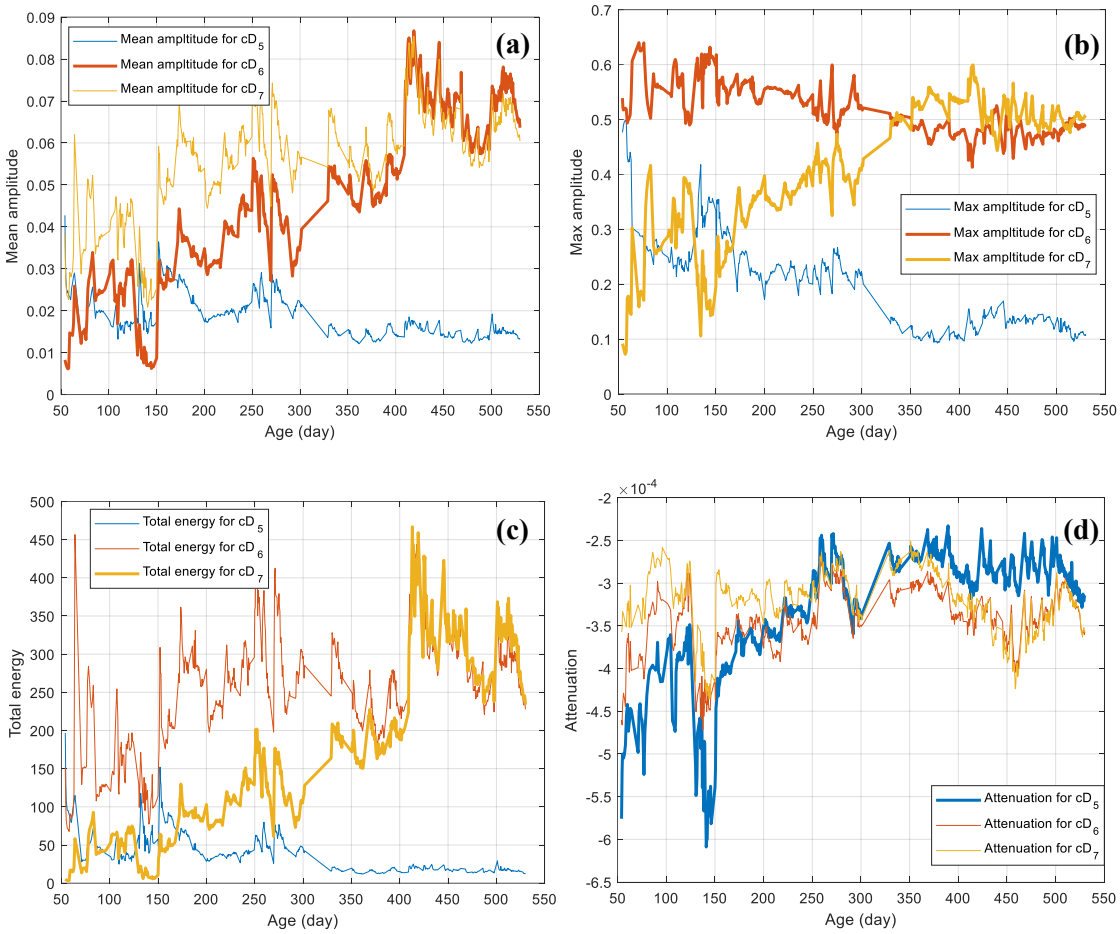

Figure 4-3. Wavelet feature histories for the ASR specimen: (a) mean amplitudes, (b) maximum amplitudes, (c) total energies, and (d) wave attenuations.

4.2 PREPROCESSING OF FULL TIME-DOMAIN SIGNAL AND FREQUENCY SPECTRUM

The full time-domain signals and frequency spectra were used as inputs for the DL models (e.g., DNNs). However, these two types of data cannot be input into the model directly. The original data need to be cleaned and downsampled to remove useless information, thus reducing the computational cost for model training and testing. Figure 4-4(a) shows the original time-domain signal collected on the ASR specimen at the age of 54 days. The time-domain signal had a total number of 36,000 data points with a sampling rate of 10 M samples/second. First, the signal was normalized to a maximum absolute amplitude of one and truncated to 7,000 samples by removing the 31,000 data points in the tail. Then, the truncated signal was downsampled with a ratio of 13, and a new signal was obtained with a total of 538 samples. The new sampling rate was 769 k samples/second after downsampling. Because the energy of the signal was in the frequency range of 50 to 200 kHz, the new sampling rate should have been sufficient to keep the original information. After downsampling, the new sampling rate should be at least higher than two times the frequency upper limit of the signal. To facilitate the model input, the first 27 data points were removed to make the final signal with 512 samples, which is shown in Figure 4-4(b).

Figure 4-5(a) shows the frequency spectrum for the corresponding time-domain signal. The main energy is in the range of 50 to 200 kHz. The information outside this frequency range is useless. Therefore, the frequency spectrum was truncated in the frequency range of 57 to 200 kHz and then downsampled with a ratio of 2. The final spectrum used for the DNN input is plotted in Figure 4-5(b) with a total of 256 data points.

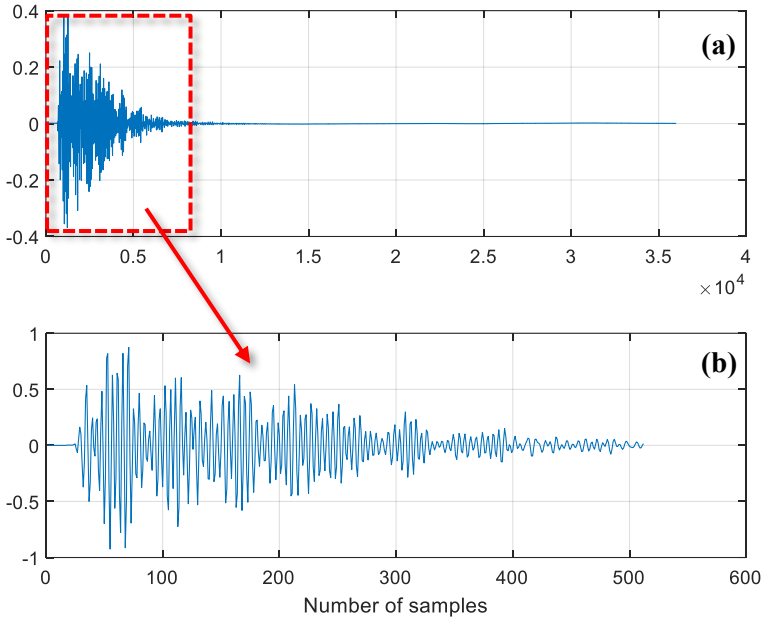

Figure 4-4. (a) Original time-domain signal from the ASR specimen, and (b) processed and downsampled signal for DNN input.

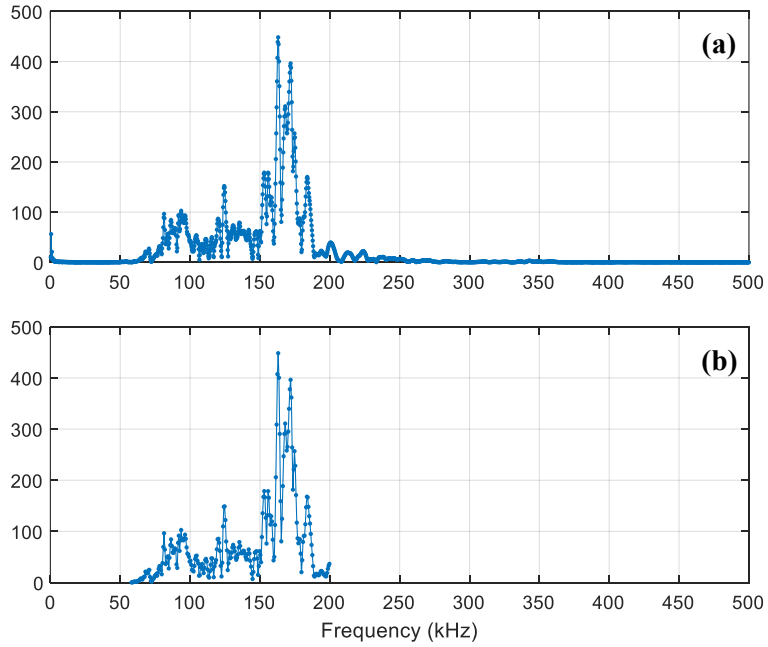

Figure 4-5. (a) Original frequency-domain spectrum from the ASR specimen, and (b) processed and downsampled spectrum for DNN input.

5. MACHINE LEARNING MODELS

In this work, two different ML models were used as the prototypic models for the ASR expansion predictions, including SVR and DNN approaches. The DNN uses the processed time-domain signals and frequency spectra as the feature input, and the SVR uses the extracted features as the model input.

SVR is an ML algorithm used for regression tasks. It is based on the principles of SVMs. An SVM is a supervised ML algorithm primarily used for classification tasks, but it can also be applied to regression tasks. The basic principle of the SVM is to find the optimal hyperplane (decision boundary) that best separates data points belonging to different classes in the feature space. For a linear model with dataset (x, y), the hyperplane $f(x)$ is expressed as

$$f(x) = wx + b = 0, \tag{5}$$

where w is the weight vector, and b is the bias term. The hyperplane should be subject to the following constraints:

$$y_i f(x_i) = y_i(wx_i + b) = 0, \tag{6}$$

whereas the cost function is

$$J = \frac{1}{2}\|w\|^2. \tag{7}$$

The constraint and the cost function are for separable data. If the data are not separable, then a soft margin is used with a cost function:

$$J = \frac{1}{2}\|w\|^2 + c\sum_{i=1}^{m}\xi_i\,, \tag{8}$$

with constraint

$$y_i f(x_i) = y_i(wx_i + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0, \tag{9}$$

where ξ_i is a positive slack variable that indicates that the sample training allows a small number of erroneous samples, and c is the penalty parameter. The SVM model has several hyperparameters: the kernel function, kernel scale, and penalty parameter. Linear kernels [41], Gaussian radial basis function [22], [42], [43], polynomials [44], [45], and sigmoid functions are among the most common kernel functions. The kernel scale defines how far the influence of a single training example reaches. The penalty parameter controls the trade-off between minimizing and maximizing the classification margin and training error.

Instead of predicting discrete class labels as in classification using SVM, SVR predicts continuous output values. Similar to SVM for classification, SVR aims to find a hyperplane in the feature space that best fits the training data while minimizing the error, also known as the *epsilon-insensitive loss function*. This hyperplane is determined by support vectors, which are the data points closest to the hyperplane and influence its position. SVR allows for flexibility in modeling nonlinear relationships between input features and output values by using kernel functions to map the input data into higher-dimensional feature spaces where linear separation may be achieved. Other hypermeters include kernel scale, Box Constraint, and epsilon, which is the tolerance used in the epsilon-insensitive loss function. These parameters for the

SVR models used in this work have been summarized in Appendix A. SVR aims to minimize the error and ensure that the deviations of the actual output values from the predicted values within the epsilon tube do not exceed a predefined threshold. This approach makes SVR robust to outliers and noisy data, making it suitable for regression tasks in various domains.

Another model used in this work is a DNN. The DNN is a type of artificial neural network with multiple layers between the input and output layers. An artificial neural network is a connected graph with multiple layers, and each layer contains multiple nodes (neurons). A shallow feed-forward neural network has a relatively simple structure, with one input layer, one to two intermediate (hidden) layers, and one output layer. Lippmann [46] suggested that a multilayer perceptron with two hidden layers is sufficient for creating classification regions of any desired shape. Another critical hyperparameter is the activation function, which defines the output of the node given the input or a set of inputs. Typical activation functions include the Gaussian, sigmoid, hyperbolic tangent, and radial basis functions. The parameters in the hidden layers can be optimized to achieve the best model performance. The optimized neural network used in this work has two hidden layers in addition to the input and output layers. The first hidden layer contains 16 neurons, and the second hidden layer has 8 neurons with *relu* activation functions for both layers. A randomly selected 25% of the training data was used as the validation data.

DNNs are characterized by their depth, meaning they have many hidden layers compared with traditional shallow neural networks, which may only have one or two hidden layers. Each layer in a DNN typically consists of multiple neurons, also known as *nodes* or *units*, which perform mathematical operations on the input data. The output of each layer serves as the input to the next layer, with the final output layer producing the prediction or classification. DNNs can learn intricate patterns and representations from complex data, making them particularly effective in tasks such as image and speech recognition, natural language processing, and other areas where the input data has a high degree of complexity. Training a DNN involves feeding it with labeled training data and adjusting the weights and biases of the connections between neurons through a process called backpropagation to minimize the difference between the predicted output and the true output. This training process often requires a large amount of data and computational resources but can result in highly accurate models capable of handling complex tasks.

In this work, time-domain signals and frequency spectra were used as the input of the DNN models. Figure 5-1 shows an example of the network structure of the DNN used herein; it has three hidden layers with *relu* activation functions. The input layer has 512 neurons if the time-domain signals are used as the input, and the second hidden layer has 128 neurons. The third and fourth hidden layers have 32 and 8 neurons, respectively. Dropout layers are added to the network to prevent overfitting of the DNN. The first and second dropout layers have a dropout rate of 0.4, whereas the last one has a rate of 0.3. If the frequency spectra are used as the model input, the number of neurons for each hidden layer will differ from those used here.

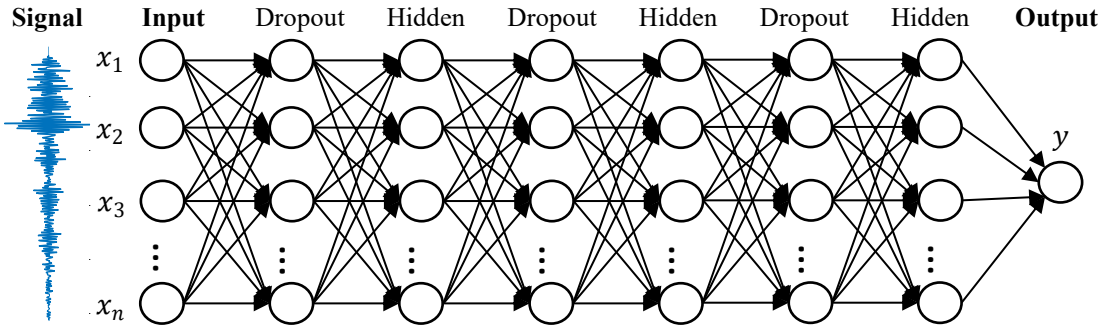

Figure 5-1. The network structure of the DNN model.

6. RESULTS

6.1 DATA RANGE OF THE TRAINING AND TESTING DATA

In the previous work [21], the SVR model trained with the data from the ASR sample showed good prediction performance on the testing data from the ASR-2D sample. The goodness of fit R^2 and RMSE are used as the two metrics to evaluate the model prediction performance. When R^2 approaches 1, and RMSE approaches 0, these parameters indicate the optimal prediction performance. The prediction results here have an R^2 of 0.894 and an RMSE of 0.057%, as shown in Figure 6-1. The reason for using the data from ASR as training and data from ASR-2D as testing is that the ASR data has a larger feature space than that of the ASR-2D. In the previous work, researchers selected six features extracted from the ultrasonic signals as the input for the ML learning models. In the present work, all six features were processed using principal component analysis (PCA) to calculate two main principal components to characterize the feature space of the ASR and ASR-2D data.

Figure 6-2 plots the feature space of the ASR and ASR-2D data using the two extracted principal components. The ASR data have a large feature space (blue rectangle), but the ASR-2D data have a much smaller feature space (red rectangle). Additionally, most of the data points of ASR-2D were inside the ASR feature space. Therefore, training with the ASR data and testing with the ASR-2D data for expansion prediction is interpolation. This study also observed that some of the ASR-2D data points are outside the ASR feature space. These data points correspond to the blue data points in Figure 6-1, where large variations between the predicted expansion and measured expansion are observed. Therefore, if the test data are outside the range of the training data, poor performance may be observed for the data points.

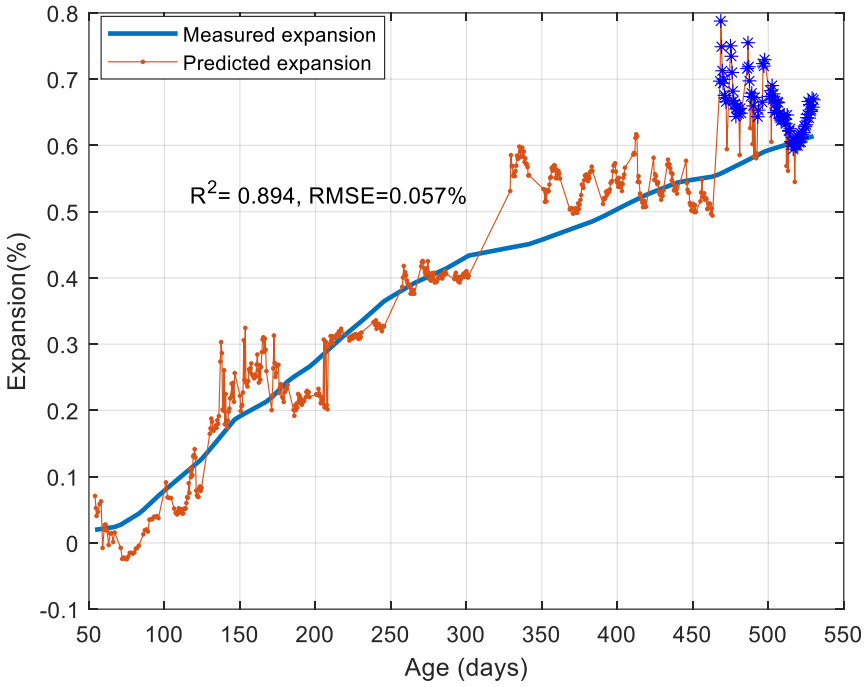

Figure 6-1. Testing results on the ASR-2D data using the SVR model trained by the ASR data.

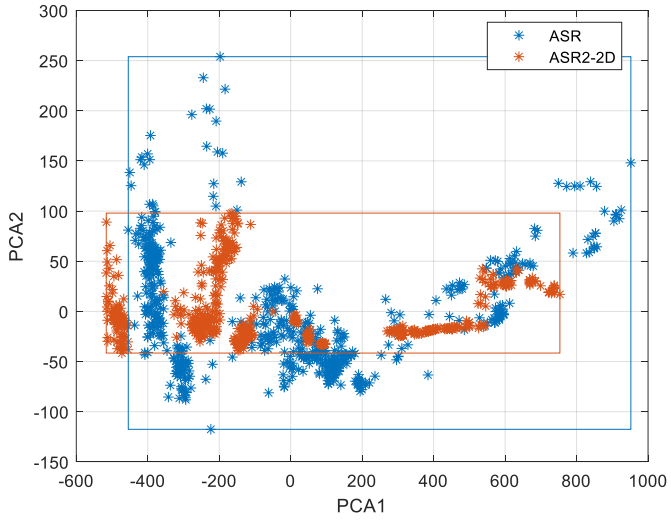

Figure 6-2. Feature space plot using two principal components extracted from the ASR and ASR-2D data.

Then, the SVM model was trained using the ASR-2D data and tested with the ASR data. The result is plotted in Figure 6-3. The model performance using ASR-2D as training data was poorer than the result in Figure 6-1 (using ASR as the training data) because the ASR-2D data had a smaller feature space or data range than that of the ASR data. The training data only had a maximum expansion of 0.6%. Figure 6-3 also shows that the predicted expansions match well with the measured expansion in the range of <0.6% measured expansion, while large variations were observed in the range of >0.6% measured expansion. The RMSE for the range <0.6% was 0.066%, and the RMSE was 0.141% in the other range. These results also verify that the testing data should be in the range of the training data to achieve good ML performance.

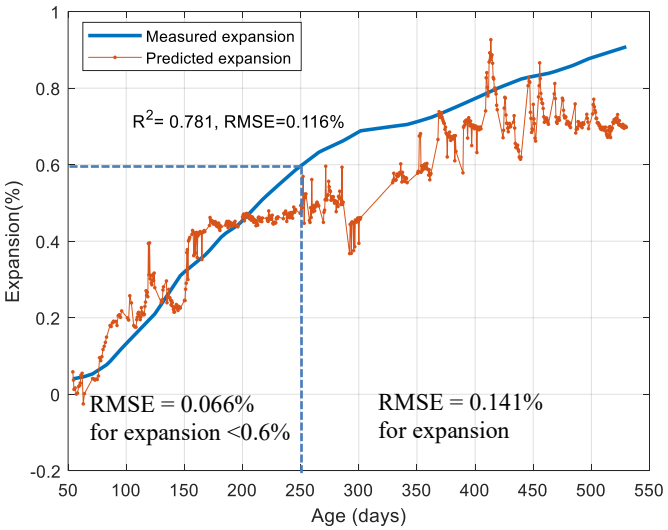

Figure 6-3. Testing results on the ASR data using the SVR model trained by the ASR-2D data.

6.2 INFLUENCE OF SIGNAL PREPROCESSING

The signal preprocessing procedure usually has a large influence on the ML performance. In this work, the effect of the normalization process on ML performance was studied. During the long-term monitoring of ASR development in concrete, the ultrasonic signal amplitude would decrease because of multiple

factors: concrete cracking, sensor coupling deterioration, temperature change, and more. Therefore, the ultrasonic signal should be normalized before the feature extraction and input to the ML models. Figure 6-4 shows the time-domain signals at 76 days, 118 days, and 163 days. The signal peak amplitude decreased from 3 V at 76 days to 0.075 V at 163 days. The peak amplitudes of all time-domain signals over the monitoring period are plotted in Figure 6-5. After 160 days, the signal amplitude became very small.

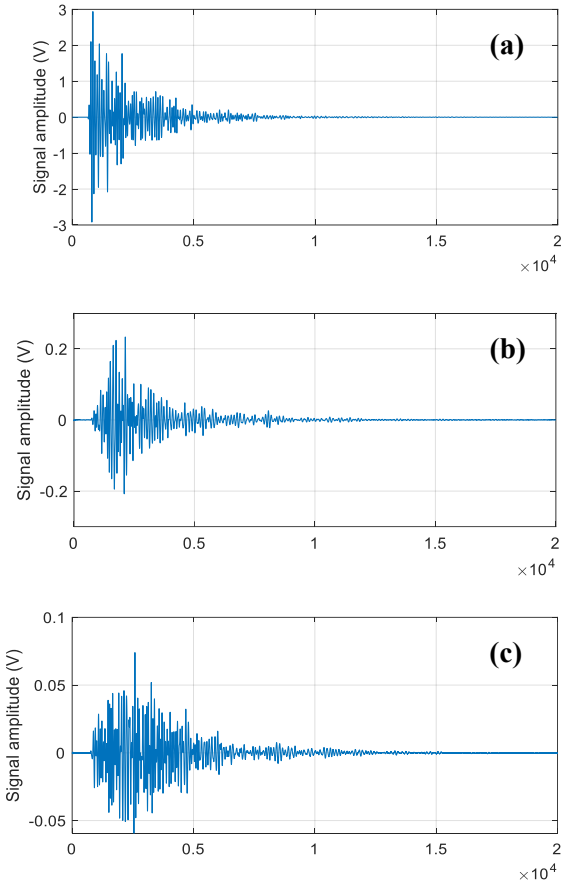

Figure 6-4. Time-domain signals at the concrete ages of (a) 76 days, (b) 118 days, and (c) 163 days.

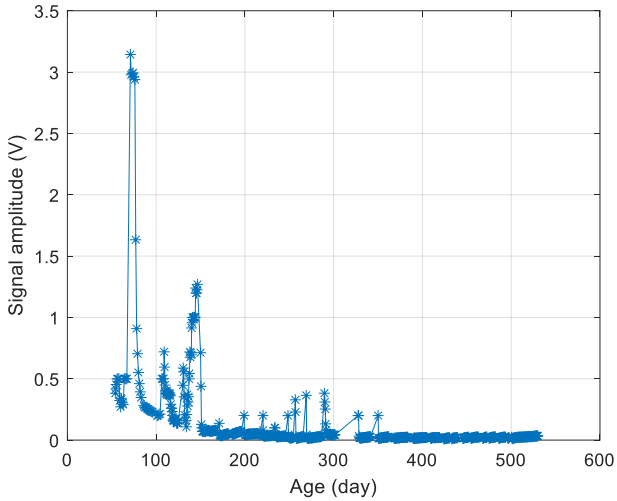

Figure 6-5. Peak amplitudes of the time-domain signals over the monitoring period.

The SVR model was first trained with the data from the ASR sample and then tested with the data from the ASR-2D sample. For both the training and testing data, original time-domain signals were used for the feature extraction without normalization. Figure 6-6 plots the testing results of this trained model using original signals without normalization. The performance was poor, with a low R^2 of 0.592 and a large RMSE of 0.112%. This result was used as the reference for the following results. For the second model, all the time-domain signals were normalized by scaling the maximum amplitude to one for the feature extraction in both the training and testing datasets. The testing results on the ASR-2D dataset are plotted in Figure 6-7. The testing performance was observed to be significantly improved with a higher R^2 of 0.894 and a smaller RMSE of 0.057%, indicating that the normalization of signal amplitudes for both training and testing can improve the model performance, especially when the features are related to the signal amplitude.

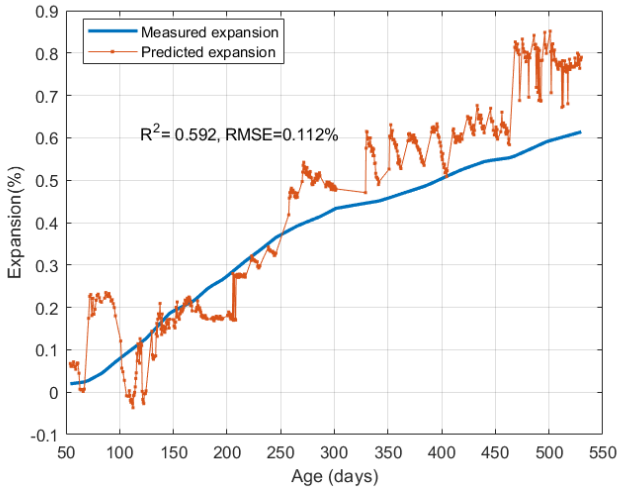

Figure 6-6. Test results using original signals for both training and testing datasets (SVR model).

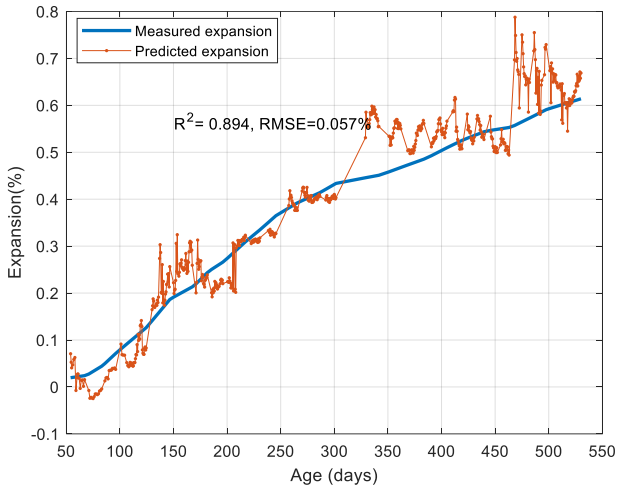

Figure 6-7. Test results using normalized signals for both training and testing datasets (SVR model).

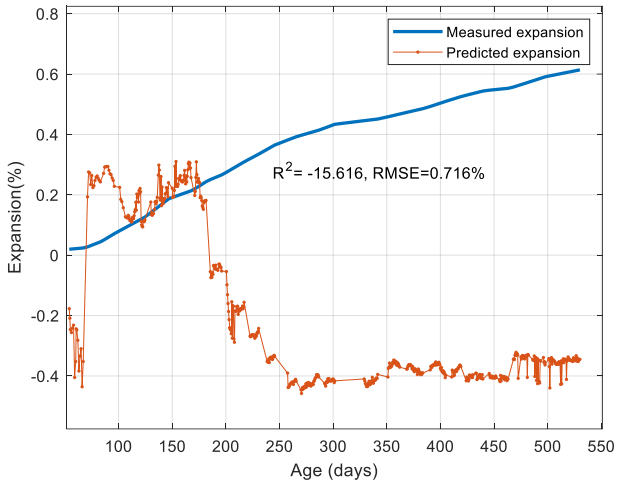

Figure 6-8. Test results using normalized signals for the training dataset and original signals (without normalization) for the testing dataset (SVR model).

In the third test, the training model was the same as the one used in Figure 6-7, which was trained using normalized signals. However, the testing data here were from the original signals without amplitude normalization. The result is shown in Figure 6-8. The prediction performance was even worse than that in Figure 6-6, using both original signals for both training and testing datasets with a negative R^2. A negative R^2 indicates a relatively poor generalization performance with the trained model on the test data. Because the training dataset was from the normalized signals here, the trained model based on this dataset would not generalize well on the dataset without normalization. The results for three scenarios are summarized in Table 6-1. Therefore, the preprocessing, such as normalization of the signal amplitudes, should be consistent for training and testing data to ensure good prediction performance.

Table 6-1. Influence of the signal normalization on the test results (SVR models)

Training data	Testing data	Testing results	
		R^2	RMSE (%)
Original ASR data	Original ASR-2D data	0.592	0.112
Normalized ASR data	Normalized ASR-2D data	0.894	0.057
Normalized ASR data	Original ASR-2D data	−15.6	0.716

6.3 INFLUENCE OF THE TESTING TEMPERATURE ON THE MODEL PERFORMANCE

In this work, all the ultrasonic signals were collected at the specimen curing temperature of 38°C. However, the ultrasonic signal was very sensitive to temperature changes. Therefore, ultrasonic signals collected at different temperatures were used as the testing data with the existing ML model trained by ultrasonic signals collected at 38°C. A group of data for the ASR and ASR-2D specimens was selected when the chamber was shut down on July 2, 2019, corresponding to the age of 217 days in Figure 3-2. At the time of the shutdown, the volumetric expansions of the ASR and ASR-2D specimens were 0.509% and 0.309%, respectively. After the shutdown, the specimen temperature dropped from 38.5°C to 23.7°C during the cooling process. Meanwhile, ultrasonic signals were collected every 10 min. The temperature history is plotted in Figure 6-9.

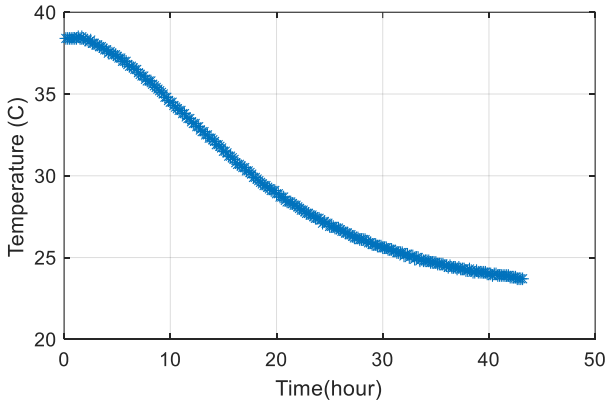

Figure 6-9. Temperature history of the ASR specimen after the chamber shut down on July 2, 2019 (217 days).

Figure 6-10 shows three time-domain signals at three different temperatures (38°C, 30°C, and 24°C) during the shutdown. In general, the waveform shapes are very similar for the three signals but with different maximum amplitudes. The temperature effect on the ultrasonic signals can be observed directly from the waveforms. The wave features were analyzed from these time-domain signals. The selected six features (see Section 3.1) at different temperatures were plotted in Figure 6-11. A relatively large temperature effect was observed for these six features in the temperature ranges of <26°C and >36°C, but these features were relatively stable in the temperature range between 26°C and 36°C.

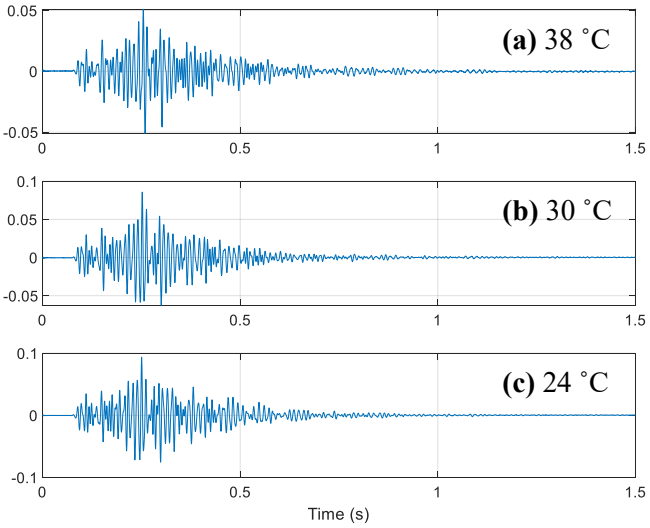

Figure 6-10. Time-domain signals on the ASR specimen at three temperatures during the shutdown: (a) 38°C, (b) 30°C, and (c) 24°C.

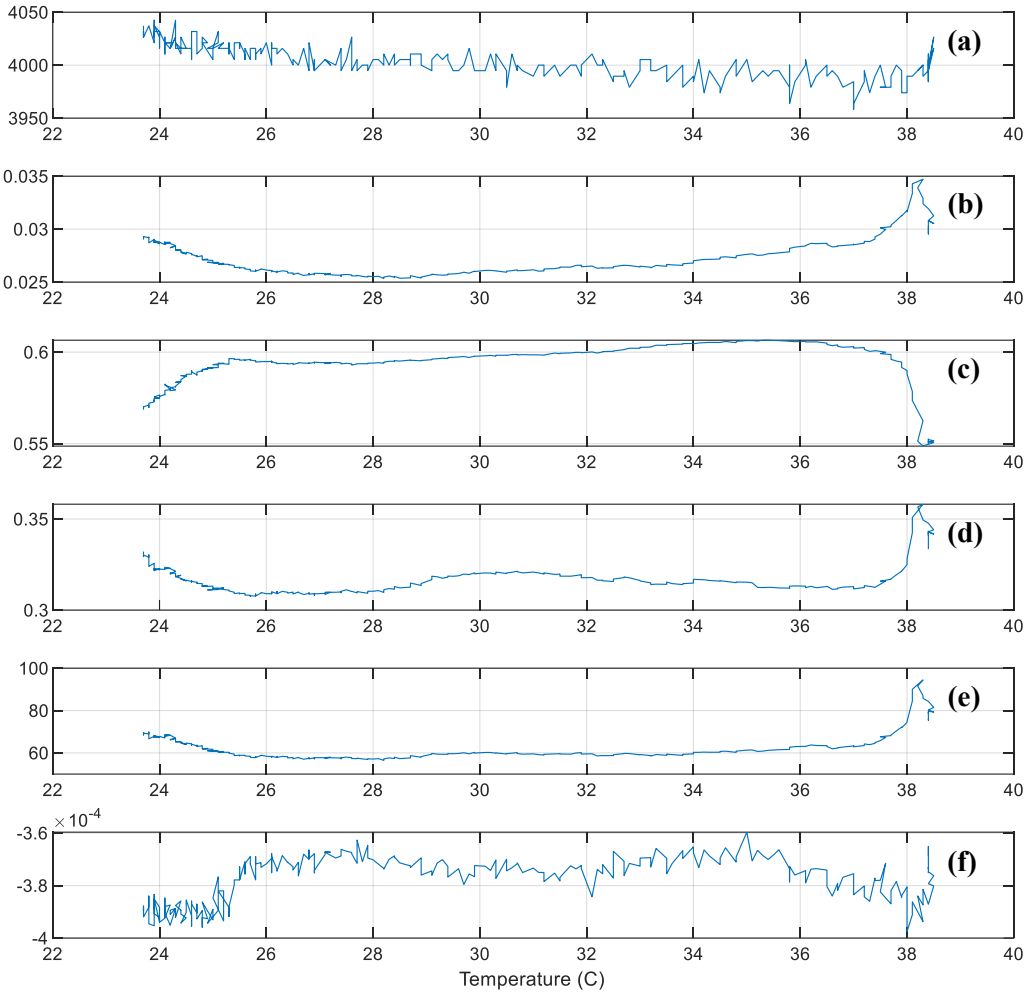

Figure 6-11. Six selected features at different temperatures from 23.7°C to 38.5°C: (a) wave velocity, (b) mean amplitude for cD_6 (78–156 kHz), (c) maximum amplitude for cD_6 (78–156 kHz), (d) maximum amplitude for cD_7 (39–78 kHz), (e) total energy for cD_7 (39–78 kHz), and (f) attenuation for cD_5 (156–312 kHz).

The test results on the ASR signals collected during the chamber shutdown are plotted in Figure 6-12(a). The measured expansion before the chamber shutdown is plotted via the red line, representing the true expansion at that time, assuming the expansion did not change during the shutdown. The predicted expansion (blue dots) was lower than the measured expansion, and the absolute prediction errors are larger than 5% in Figure 6-12(b), even though the training data and testing were collected from the same specimen. Because the training data was collected at 38°C, the prediction error at 38°C was also the smallest for this group of data compared with the data at other temperatures. The test results on the ASR-2D data were plotted in Figure 6-13(a) with blue dots for the predicted expansion and red lines for the measured expansion. The predicted expansions for temperatures below 34°C were close to the measured expansion, and large prediction errors were observed in the range >34°C in Figure 6-13(b). In this case, the testing and training data were from two different specimens. Therefore, the prediction errors at various temperatures are different from those from the ASR specimen in Figure 6-12(b).

Based on the two prediction results at different temperatures, this study can conclude that the temperature effect on the ASR expansion prediction based on ultrasonic features cannot be neglected, especially when the training data and testing are collected at different temperatures with a large temperature difference (>5°C). The temperature change will influence the crack opening and closing, thus changing the

ultrasonic wave transmitting through these cracks. Ultrasonic signals collected at different temperatures may contain slightly different information about the concrete damage. This study suggests using training and testing data that are collected at the same temperature. Another solution might be using training data that are collected at a large range of temperatures. So, the temperature of the testing data can fall into the range of the training data.

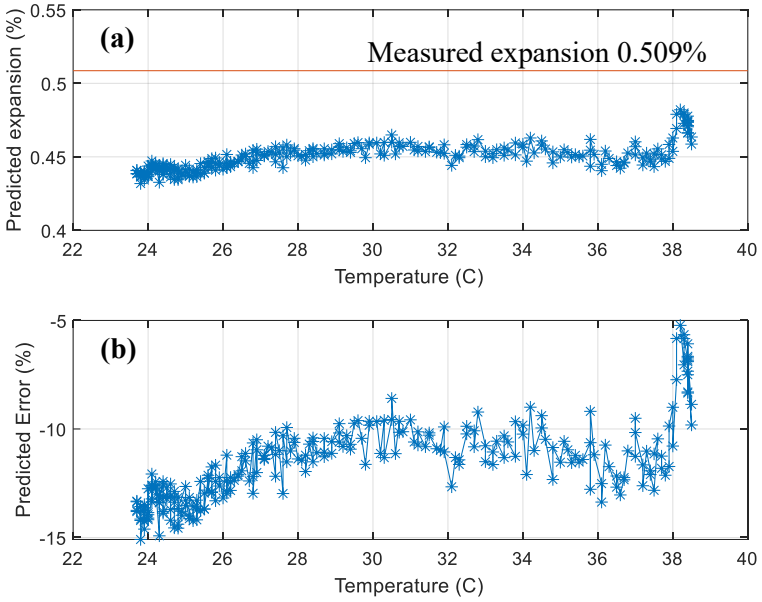

Figure 6-12. Test results of the signals collected on the ASR specimen (during shutdown) using the SVR model trained by ASR signals: (a) predicted expansions at different temperatures and (b) relative errors compared with the measured expansion.

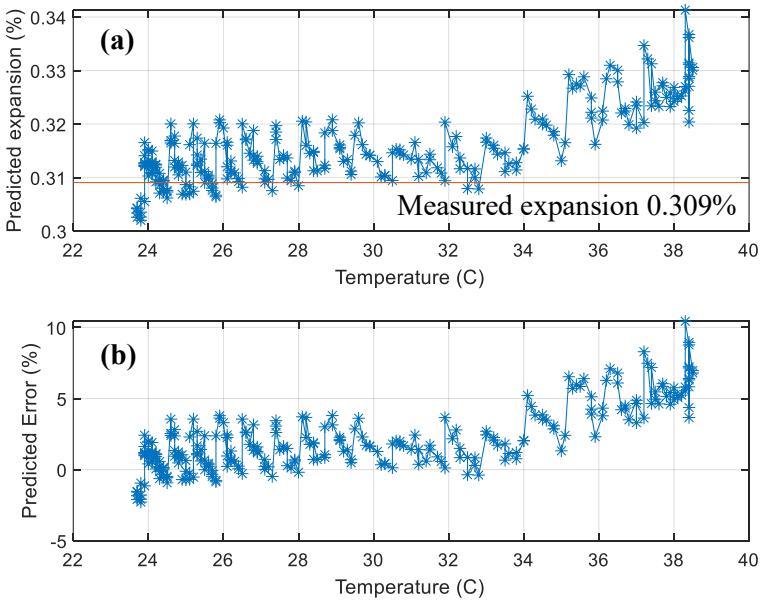

Figure 6-13. Test results of the signals collected on the ASR-2D specimen (during shutdown) using the SVR model trained by ASR signals: (a) predicted expansions at different temperatures and (b) relative errors compared with the measured expansion.

The ASR data collected during the chamber shutdown was also tested using the DNN model trained with the ASR time-domain signals. Therefore, processed time-domain signals at different temperatures were also used for the testing data. The predicted expansion was plotted in Figure 6-14, along with the measured expansion of 0.509% (red line). First, the predicted expansions at different temperatures using the DNN model and time-domain signals showed smaller variations along the temperature axis than those using the features and SVR model in Figure 6-12. Additionally, the predicted expansions were closer to the measured expansion with smaller errors than the results in Figure 6-12. This result indicates that the time-domain signals might be less sensitive to the temperature change as the model input compared with the extracted wave features. More studies are needed to verify this preliminary conclusion about the effect of temperature on ML model performance for expansion prediction.

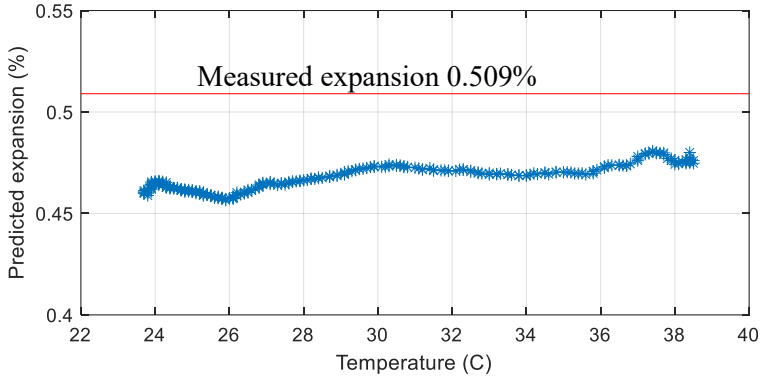

Figure 6-14. Test results of the signals collected on the ASR specimen (during shutdown) using the DNN model trained by the ASR data.

6.4 TRAINING AND TESTING USING DATA FROM DIFFERENT BATCH SPECIMENS

In this section, the ML models were trained and tested using data from different batches of specimens, such as training with the ASR data but testing with the small ASR and small ASR-2D datasets. This configuration aimed to study the generalization performance of the ML model if the training and testing data were from different physical specimens. The feature space using PCA1 and PCA2 was plotted in Figure 6-15 for the ASR, ASR-2D, small ASR, and small ASR-2D datasets. In the figure, ASR and ASR-2D have larger feature spaces than those of the small ASR and small ASR-2D datasets because the two small specimens showed smaller final volumetric expansions than the ASR and ASR-2D specimens. However, seven data points (yellow points in the lower left of the figure) were also observed from the small ASR dataset that are outside the feature space of the ASR dataset (blue rectangle).

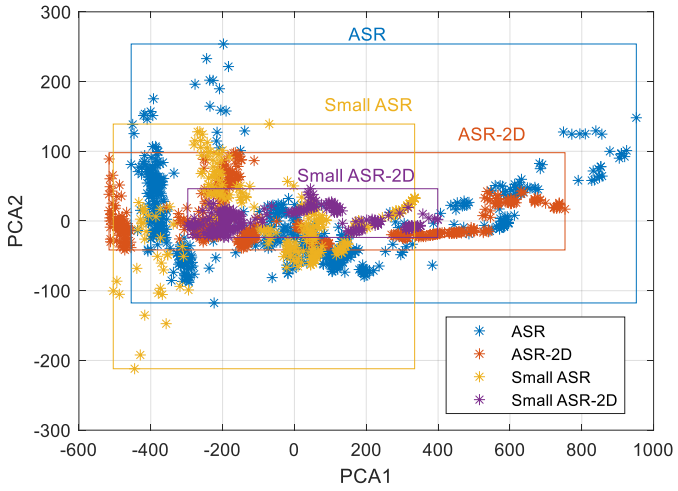

Figure 6-15. PCA space plot using two principal components extracted from the ASR, ASR-2D, small ASR, and small ASR-2D.

The SVR model was trained with the ASR data and then tested with the data of small ASR and small ASR-2D specimens. The results are plotted in Figure 6-16. The prediction performance for the two specimens is good, with similar values of R^2 and RMSE, indicating the SVR model trained with one batch of specimens can generalize well on the data collected from another batch of specimens for ASR prediction based on the ultrasonic data, regardless of the specimen size. Because the small ASR and small ASR-2D data shared many similarities, the prediction performances were also similar here using the same trained model. However, for both prediction results in the figure, the prediction expansions showed large variations with the measured expansions after 175 days (green rectangles) for both specimens. Further studies are needed for this poor performance. A possible explanation is that the ultrasonic signals were relatively weak in the late curing stage, which may have created large errors for both the training and testing data presented here.

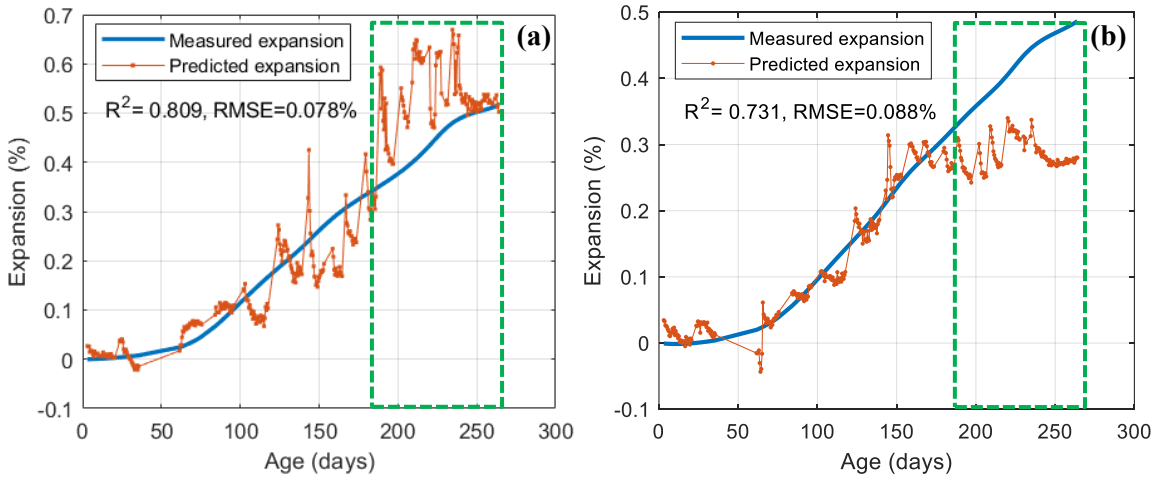

Figure 6-16. Testing results using the SVR model trained with ASR data and tested on (a) small ASR data and (b) small ASR-2D data.

The SVR model was then trained with the small ASR data [see Figure 6-17(a)] and tested with small ASR-2D data. Good prediction performance was observed on the testing data of small ASR-2D in Figure

6-17(b) with an R^2 of 0.869 and RMSE of 0.061%. Along with the previous results using the ASR data for training and ASR-2D for testing, this study can conclude that the proposed SVR model with feature selection can achieve good performance on the expansion prediction if training and testing are from the same batch of specimens. Then, the same model was tested using the ASR data and ASR-2D data, and the results are plotted in Figure 6-18. The overall prediction of the ASR data was poor, with large errors and a small R^2 in Figure 6-18(a). However, the prediction for expansion <0.56% was still acceptable, and the prediction for expansion >0.56% showed large errors compared with the measured expansion. A possible reason is that the training data had a maximum expansion of 0.52%. Therefore, the prediction performance will be poor for predictions outside this range. The prediction on the ASR-2D data showed a good performance with high R^2 and small RMSE since the training data (from a small ASR specimen) and testing data (from an ASR specimen) had similar maximum expansion (around 0.61%). This result further confirms the necessity for the training data to encompass a comparable or broader range of data points compared to the testing data, thereby validating the conclusion.

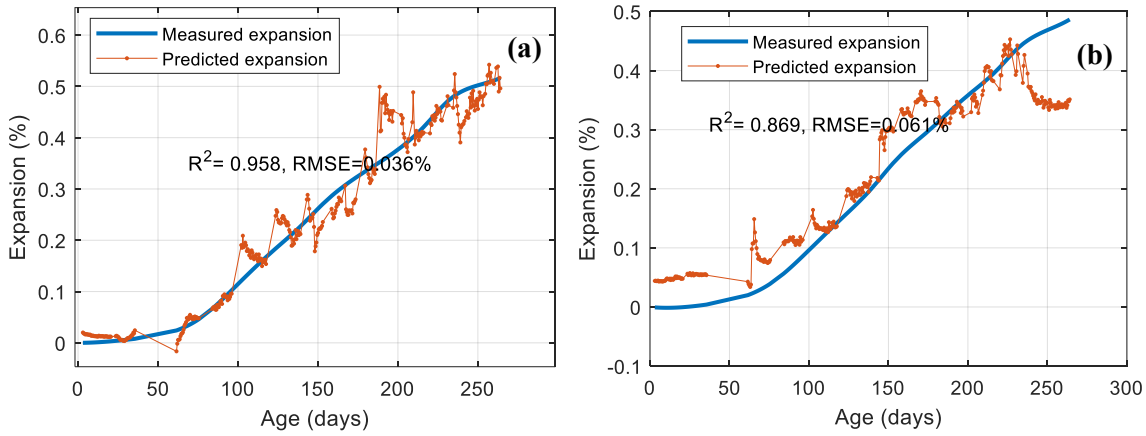

Figure 6-17. SVR model trained with small ASR data: (a) training results and (b) testing results on the small ASR-2D data.

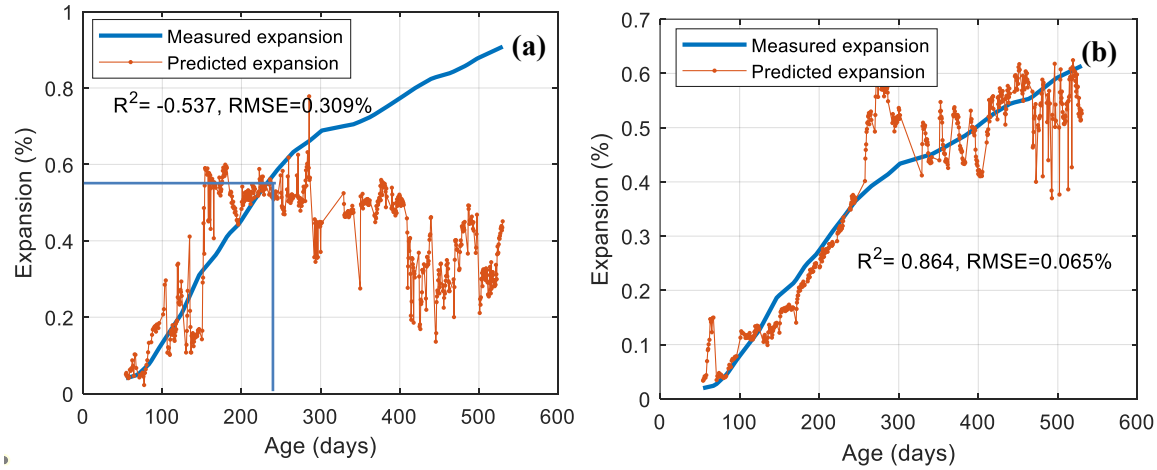

Figure 6-18. SVR model trained with small ASR data and (a) testing results on the ASR data and (b) testing results on the ASR-2D data.

6.5 USING TIME-DOMAIN SIGNAL AND FREQUENCY SPECTRUM AS THE DNN INPUT

In this section, processed time-domain signals and frequency spectra were used as both training and testing datasets for a DL model—specifically, DNNs. Initially, the DNN model was trained using time-domain signal data from the ASR specimen and subsequently tested on the ASR-2D dataset. The training phase demonstrated promising results with a high R^2 value and minimal RMSE, indicative of a well-optimized model in Figure 6-19(a). However, when tested on the ASR-2D dataset, although the predicted expansion aligns well with measured expansion up to 380 days, significant prediction errors emerge thereafter [see Figure 6-19(b)]. This behavior exists across multiple training and testing iterations, resulting in subpar predictions during the later stages of the monitoring period.

Subsequently, the DNN model is evaluated using data from two smaller specimens: small ASR and small ASR-2D datasets. The testing procedures mirror those of the training phase, using 512 data points as input. For the small ASR specimen [Figure 6-20(a)], poor prediction performance and substantial errors persisted throughout the monitoring period. Conversely, for the small ASR-2D specimen [Figure 6-20(b)], satisfactory predictions were observed between 140 and 230 days; however, overall performance remains unsatisfactory, characterized by notable prediction errors in other time intervals. Consequently, the DNN model trained on time-domain signals from ASR specimens failed to generalize effectively to the datasets from the two smaller ASR specimens.

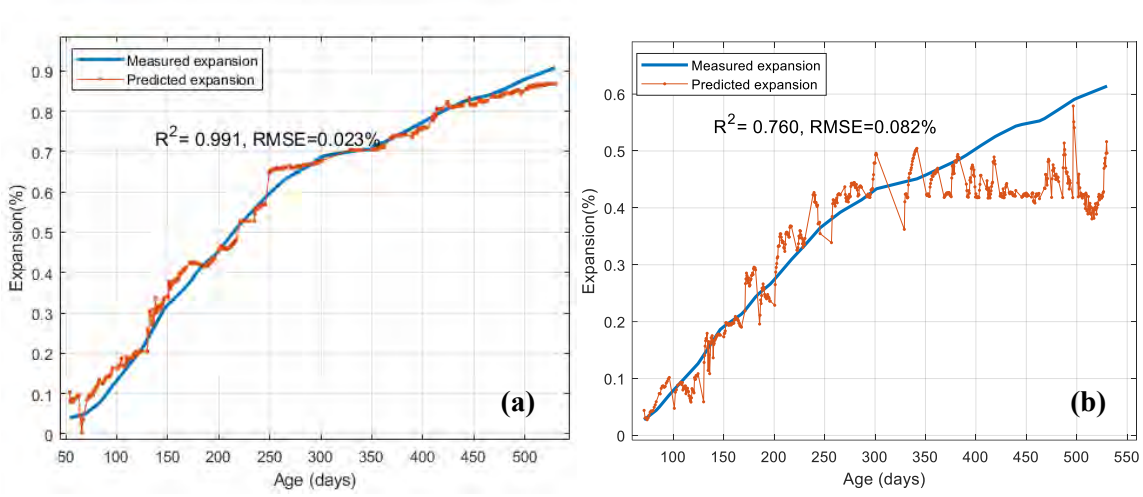

Figure 6-19. DNN model trained with ASR time-domain signals: (a) training results and (b) testing results on the ASR-2D data.

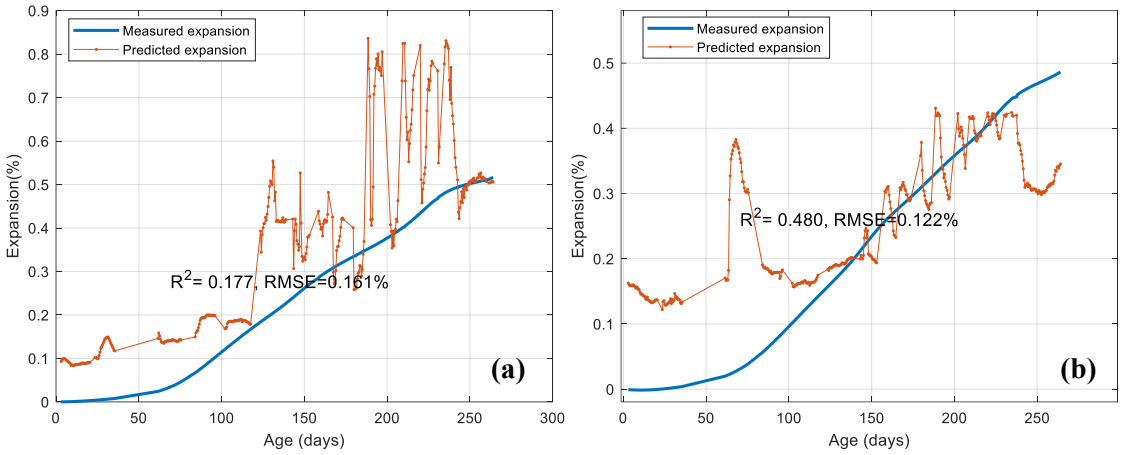

Figure 6-20. DNN model trained with ASR time-domain signals and tested on (a) small ASR data and (b) small ASR-2D data.

The DNN model was trained using time-domain signals from the ASR-2D dataset and subsequently tested using data from other specimens. The outcomes, alongside those of the model trained with ASR data, are summarized in Table 6-2. Notably, the model trained with ASR-2D data exhibited modest R^2 values and significant prediction errors. Results for models trained with small ASR data and small ASR-2D data are also presented in the table, revealing mainly poor performance characterized by low or negative R^2 values and large RMSE values. However, a noteworthy exception was observed: the model trained with small ASR-2D data demonstrated good predictive capability on the small ASR test data, achieving an R^2 of 0.795 and an RMSE of 0.081%. Visual representations of these training and testing combinations are provided in Appendix B. These results collectively suggest that the DNN models trained in this study exhibited relatively weak generalization performance. One potential contributing factor to the subpar prediction results across various training and testing combinations could be inadequate training of the DNN model and incomplete cleaning of the time-domain signal. Further investigation is needed to refine the model's performance and identify factors contributing to the observed prediction discrepancies.

Table 6-2. Summary of the results from DNN models trained with time-domain signals.

Test	Training data	Testing data	Testing results	
			R^2	RMSE (%)
1	ASR	ASR-2D	**0.760**	**0.082**
2		Small ASR	0.177	0.161
3		Small ASR-2D	0.480	0.122
4	ASR-2D	ASR	−0.128	0.265
5		Small ASR	0.559	0.119
6		Small ASR-2D	0.501	0.120
7	Small ASR	ASR	−1.142	0.365
8		ASR-2D	0.243	0.152
9		Small ASR-2D	0.425	0.129
10	Small ASR-2D	ASR	−1.843	0.421
11		ASR-2D	−0.074	0.182
12		Small ASR	**0.795**	**0.081**

The DNN model was trained and tested using frequency spectra as input data. Initially, the model was trained with frequency spectra from the ASR specimen and evaluated using data from three other specimens. The results, including training outcomes, are depicted in Figure 6-21. Notably, significant prediction errors were observed in the testing phase across ASR-2D, small ASR, and small ASR-2D specimens [refer to Figure 6-21 (b)–(d)], indicating poor generalization performance of the model trained with ASR specimen frequency spectra. These findings align with those obtained from the DNN model trained with time-domain signals, as illustrated in Figure 6-20, reflecting consistently poor prediction accuracy across models trained with different specimens. A comprehensive summary of these results is presented in Table 6-3.

However, among these outcomes, the model trained with small ASR data exhibited high prediction accuracy on the small ASR-2D test data (see Figure 6-22, Test 9 in Table 6-3). Conversely, the model trained with small ASR-2D data demonstrated good prediction results on the small ASR dataset (Figure 6-23, Test 12 in Table 6-3). This phenomenon can be attributed to the striking similarities between the two datasets (small ASR and small ASR-2D) because of their shared origin in the same batch and identical data collection setups. Moreover, these instances underscore the potential of DNN models to achieve robust generalization performance when leveraging frequency spectra as input data. Nonetheless, further refinement efforts are required to enhance overall model performance.

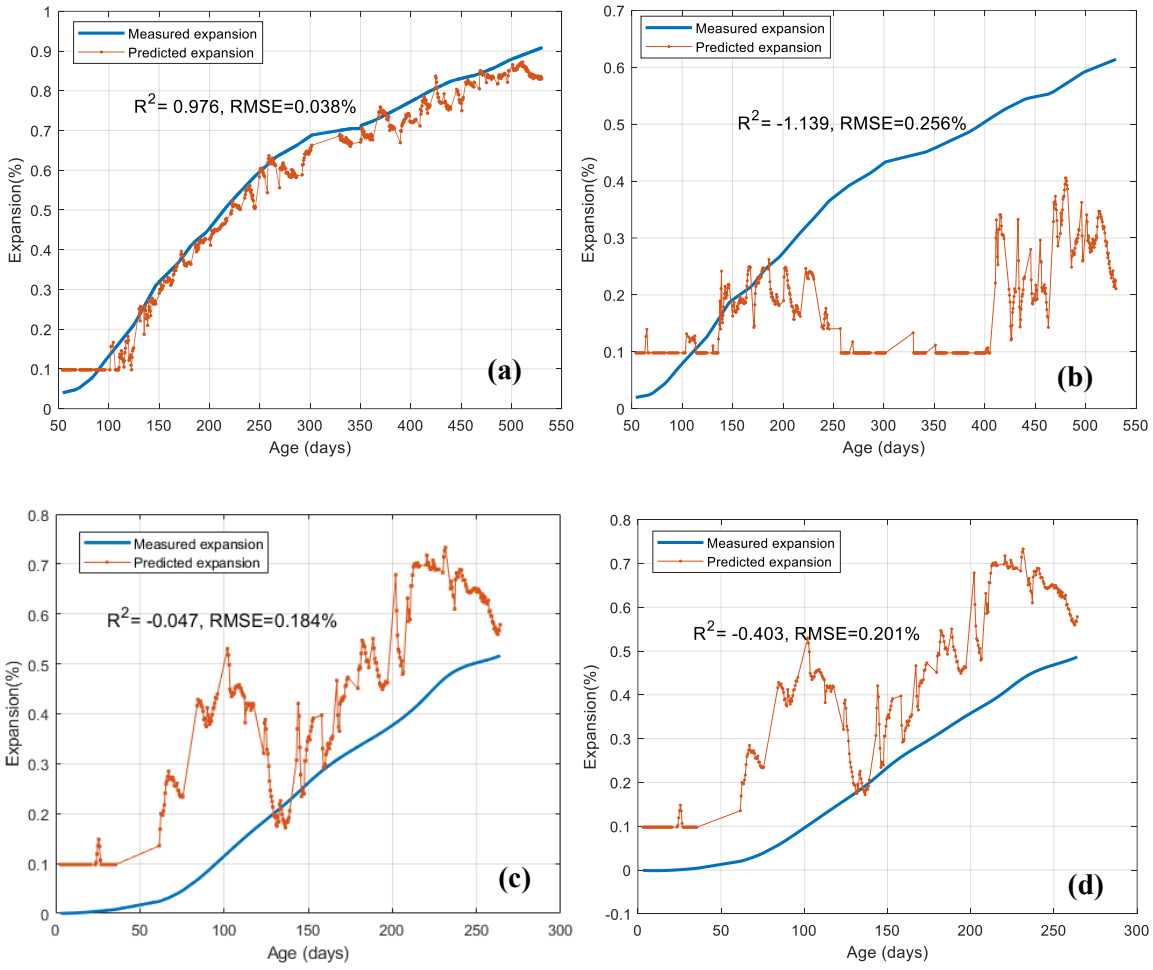

Figure 6-21. DNN model trained with ASR frequency-domain spectra: (a) training results, (b) testing results on the ASR-2D data, (c) testing results on small ASR data, and (d) testing results on small ASR-2D data.

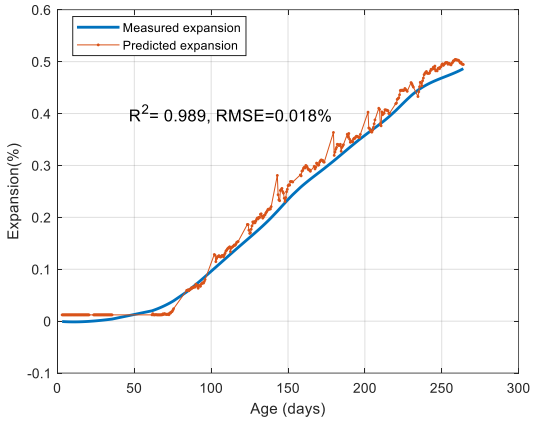

Figure 6-22. DNN model trained with small ASR frequency-domain spectra and tested on small ASR-2D data.

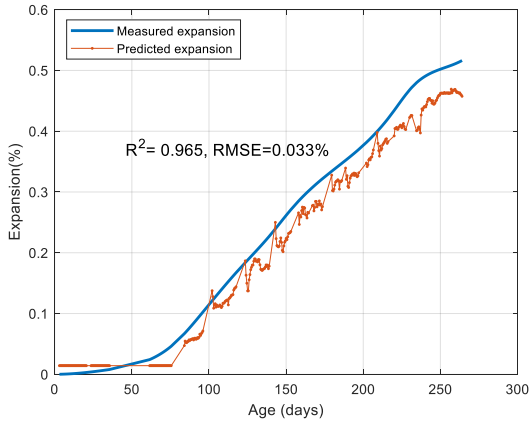

Figure 6-23. DNN model trained with small ASR-2D frequency-domain spectra and tested on small ASR data.

Table 6-3. Summary of the results from DNN models trained with frequency spectra.

Test	Training data	Testing data	Testing results	
			R^2	RMSE (%)
1	ASR	ASR-2D	−1.139	0.256
2		Small ASR	−0.047	0.184
3		Small ASR-2D	−0.403	0.201
4	ASR-2D	ASR	−0.801	0.334
5		Small ASR	−0.044	0.183
6		Small ASR-2D	−0.366	0.198
7	Small ASR	ASR	−2.010	0.433
8		ASR-2D	−2.688	0.336
9		Small ASR-2D	**0.989**	**0.018**
10	Small ASR-2D	ASR	−2.112	0.440
11		ASR-2D	−2.814	0.342
12		Small ASR	**0.965**	**0.033**

7. SUMMARY

In this report, two different ML regression models were assessed to predict the concrete material damage induced by the ASR expansion based on the long-term monitoring of ultrasonic data. The first model was a shallow ML model, SVR, that used the feature extracted from the ultrasonic signals as the model input. The second model was a DNN model that used processed time-domain ultrasonic signals and frequency spectra as the model input. In the feature engineering for the SVR model, six features were selected from the total 13 extracted features. The feature extraction and feature selection processes were discussed in the previous report [21]. For the DNN model, the time-domain signal and frequency spectrum were first truncated and downsampled to remove useless information and reduce computational cost before input into the DNN models. Data from four concrete specimens with artificial ASR were studied using the two ML models. Multiple factors that influence the prediction accuracy and errors of the regression models were investigated.

The results to date indicate a potential for wide variation in performance with ML algorithms capable of achieving very high or very low classification accuracy. Collectively, these results appear to suggest the following.

- The data range of the training and testing data was analyzed using PCA. Results indicated that the data range of the testing dataset should be inside the data range of the training dataset. This did not guarantee good prediction accuracy on the testing data. However, poor prediction results were observed if the test data range was outside the training dataset range. Therefore, the training dataset should be carefully selected and well-characterized to ensure its data range is large enough to cover the data range of the testing dataset. Datasets used for training should be diverse and representative of the types of data expected to be encountered during use (testing) (Section 6.1).

- Preprocessing is suggested to ensure good quality for both the training and testing data. Normalization may be needed if the data do not have a consistent characteristic, such as varying amplitude of the ultrasonic signals over the monitoring period. Preprocessing procedures should be consistent across the data used for training and testing datasets. Inconsistencies in these stages can majorly affect classification accuracy (Section 6.2).

- Ultrasonic data are easily affected by many factors, such as the testing temperature and the specimen stress. These factors will influence the features extracted from the ultrasonic signals. Therefore, these effects should be considered when preparing the training and testing data, such as using temperature compensation techniques. Using temperature effect as an example, the training data should be collected at similar temperatures to that of the testing data, or the training dataset should have data collected at a large temperature range, and the temperature of the testing dataset should be included inside the range (Section 6.3).

- For ultrasonic data collected from different specimens, assuming the specimens have the same mix design, the ML models trained with data from one specimen are able to predict the ASR expansion on another specimen with an acceptable accuracy based on the ultrasonic signals. However, as mentioned in Section 6.1, poor performance will be obtained if the testing data are out of the data range of the training data. Another factor to ensure good performance is that the data collected from two different specimens should be from the same or similar transducers (e.g., similar center frequency, same mode) (Section 6.4).

- Time-domain signals and frequency spectra were used as the model input for the DNN models. However, most results did not show good prediction accuracy, and the prediction errors were relatively large. Although the input data (time-domain signals and frequency spectra) were cleaned before input, the mode trained based on these data did not seem well-trained and did not generalize well on the testing data from a different specimen. However, several exceptional cases were observed with high prediction accuracy and small errors. Therefore, the DNN model still has the potential to achieve high regression performance using time-domain signals and frequency spectra as the inputs. Meanwhile, more studies are needed for model tuning and data preprocessing to improve the model performance. An obvious conclusion from these results is that using DL models such as DNN usually requires more effort in model tuning and data processing (Section 6.5).

- Based on knowledge gained from the literature review and the work in this report, a comprehensive overview diagram of the use of ML for ultrasonic NDE is summarized in Figure 7-1. Although the diagram targeted the application of ML in evaluating concrete material damage status based on the ultrasonic NDE data, it can be applied to a generic application of using ML for ultrasonic NDE. The diagram includes seven distinct steps: measurement (input), preprocessing, feature extraction, feature selection, ML modeling, model optimization, and verification and validation. Among these steps, the preprocessing, feature selection, and model optimization are optional. Although optional, it is imperative to acknowledge the significance of these three steps in enhancing the performance of ML models for ultrasonic NDE. Thus, these steps are recommended for optimal outcomes in ML for ultrasonic NDE applications. Under each corresponding step, detailed algorithms or methods are listed to guide the implementation. Notably, some of these algorithms or methods have been employed in this study. However, it is essential to recognize that the listed algorithms or methods are merely exemplary and not exhaustive. The diagram presented herein primarily focuses on using ultrasonic time-domain signals as input. Nonetheless, the outlined procedures are also adaptable to other forms of ultrasonic data, such as B-scan or C-scan.

- The analyses presented in this study did not explicitly address the uncertainty associated with the prediction outcomes. It is essential to acknowledge that ML models generate results based on input data, but assessing their accuracy becomes challenging in real-world scenarios where the true outcomes are unknown. Therefore, implementing methodologies that estimate the confidence level of prediction results would greatly facilitate the interpretation of ML model outputs.

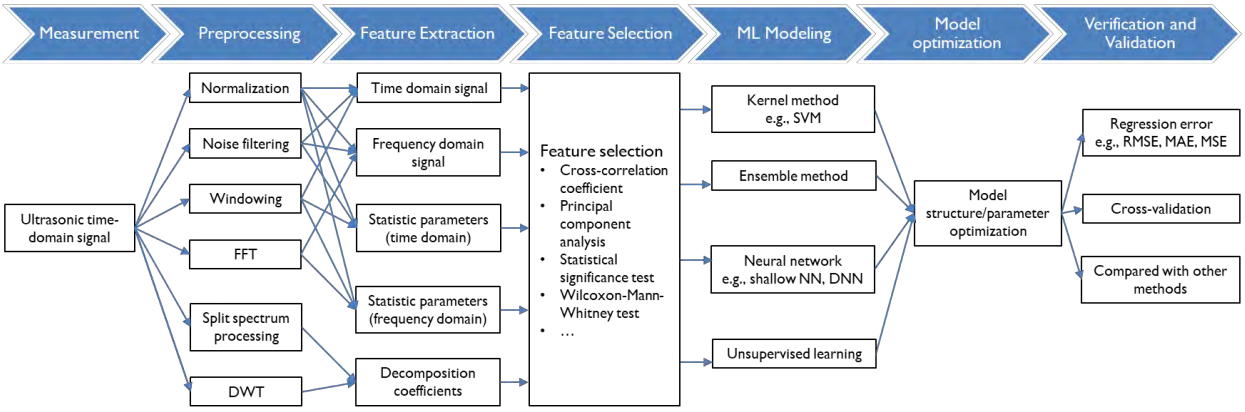

Figure 7-1. A generic ML diagram for ultrasonic NDE based on time-domain signals.

ACKNOWLEDGMENT

This research was sponsored by the US Department of Energy (DOE) Office of Nuclear Energy's Light Water Reactor Sustainability program under contract DE-AC05-00OR22725 with UT Battelle LLC/Oak Ridge National Laboratory (ORNL). The experimental ultrasonic data were shared by the University of Nebraska-Lincoln (supported by the U.S. NEUP). Xiang (Frank) Chen and Elena Tajuelo Rodriguez are also acknowledged for their invaluable efforts in project management and their insightful feedback on earlier drafts of this report.

REFERENCES

[1] M. Kawamura and K. Iwahori, "ASR gel composition and expansive pressure in mortars under restraint," *Cement and Concrete Composites*, vol. 26, no. 1, pp. 47–56, Jan. 2004.

[2] J. Zhu *et al.*, "Online Monitoring System for Concrete Structures Affected by Alkali-Silica Reaction," Univ. of Nebraska, Lincoln, NE (United States), DOE-UNL-NE8544, Dec. 2021. doi: 10.2172/1838356.

[3] B. Fournier, M.-A. Berube, K. J. Folliard, and M. Thomas, "Report on the Diagnosis, Prognosis, and Mitigation of Alkali-Silica Reaction (ASR) in Transportation Structures," Art. no. FHWA-HIF-09-001, Jan. 2010.

[4] F. Saint-Pierre, P. Rivard, and G. Ballivy, "Measurement of alkali–silica reaction progression by ultrasonic waves attenuation," *Cement and Concrete Research*, vol. 37, no. 6, pp. 948–956, Jun. 2007.

[5] E. R. Giannini, "Evaluation of concrete structures affected by alkali-silica reaction and delayed ettringite formation," thesis, 2012.

[6] T. Ju, J. D. Achenbach, L. J. Jacobs, M. Guimaraes, and J. Qu, "Ultrasonic nondestructive evaluation of alkali–silica reaction damage in concrete prism samples," *Mater Struct*, vol. 50, no. 1, p. 60, Aug. 2016.

[7] H. Sun, Y. Tang, C. Malone, and J. Zhu, "Long-term ultrasonic monitoring of concrete affected by alkali-silica reaction," *Structural Health Monitoring*, p. 14759217231169000, Apr. 2023, doi: 10.1177/14759217231169000.

[8] H. Song, S. B. Feldman, and J. S. Popovics, "In situ detection and characterization of alkali-silica reaction damage in concrete using contactless ultrasonic wavefield imaging," *Cement and Concrete Composites*, vol. 133, p. 104661, Oct. 2022, doi: 10.1016/j.cemconcomp.2022.104661.

[9] C. Malone, J. Zhu, J. Hu, A. Snyder, and E. Giannini, "Evaluation of alkali–silica reaction damage in concrete using linear and nonlinear resonance techniques," *Construction and Building Materials*, vol. 303, p. 124538, 2021.

[10] C. Malone, H. Sun, and J. Zhu, "Nonlinear Impact-Echo Test for Quantitative Evaluation of ASR Damage in Concrete," *Journal of Nondestructive Evaluation*, vol. 42, no. 4, p. 93, Oct. 2023, doi: 10.1007/s10921-023-01003-2.

[11] K. Amini, M. Jalalpour, and N. Delatte, "Advancing concrete strength prediction using non-destructive testing: Development and verification of a generalizable model," *Construction and Building Materials*, vol. 102, pp. 762–768, Jan. 2016.

[12] M. A. Kewalramani and R. Gupta, “Concrete compressive strength prediction using ultrasonic pulse velocity through artificial neural networks,” *Automation in Construction*, vol. 15, no. 3, pp. 374–379, May 2006.

[13] D. Feng *et al.*, “Machine learning-based compressive strength prediction for concrete: An adaptive boosting approach,” *Construction and Building Materials*, vol. 230, p. 117000, Jan. 2020.

[14] G. Trtnik, F. Kavčič, and G. Turk, “Prediction of concrete strength using ultrasonic pulse velocity and artificial neural networks,” *Ultrasonics*, vol. 49, no. 1, pp. 53–60, Jan. 2009.

[15] D. A. Clayton, H. Santos-Villalobos, N. D. B. Ezell, J. Clayton, and J. Baba, “Automated Detection of Alkali-Silica Reaction in Concrete Using Linear Array Ultrasound Data,” in *Proceedings of the 18th International Conference on Environmental Degradation of Materials in Nuclear Power Systems – Water Reactors*, J. H. Jackson, D. Paraventi, and M. Wright, Eds., in The Minerals, Metals & Materials Series. Cham: Springer International Publishing, 2019, pp. 1335–1345. doi: 10.1007/978-3-030-04639-2_87.

[16] L. Ai, V. Soltangharaei, and P. Ziehl, “Evaluation of ASR in concrete using acoustic emission and deep learning,” *Nuclear Engineering and Design*, vol. 380, p. 111328, Aug. 2021, doi: 10.1016/j.nucengdes.2021.111328.

[17] S. A. Miele, “Vibro-Acoustic Testing and Machine Learning for Concrete Structures Damage Diagnosis,” Thesis, 2022. Accessed: Feb. 20, 2024. [Online]. Available: https://ir.vanderbilt.edu/handle/1803/17900

[18] S. Miele, P. M. Karve, S. Mahadevan, and V. Agarwal, “Diagnosis of internal cracks in concrete using vibro-acoustic modulation and machine learning,” *Structural Health Monitoring*, vol. 21, no. 5, pp. 1973–1991, Sep. 2022, doi: 10.1177/14759217211047901.

[19] T. Shin, H. Sun, J. Zhu, and Y. Zhang, “Nondestructive Damage Detection of Concrete With Alkali-Silica Reactions Using Coda Wave and Anomaly Detection,” *IEEE Sensors Journal*, vol. 22, no. 6, pp. 6124–6135, Mar. 2022, doi: 10.1109/JSEN.2022.3149721.

[20] H. Sun, J. Zhu, and P. Ramuhalli, “Machine Learning of Ultrasonic Data for Expansion Prediction of Concrete with Alkali-Silica Reaction,” in *STRUCTURAL HEALTH MONITORING 2021*, 2021. doi: 10.12783/shm2021/36321.

[21] H. Sun and S. Sabatino, “Machine Learning for Processing Ultrasonic Data from Long-Term Monitoring of Concrete with Alkali-Silica Reaction (ASR),” Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), ORNL/SPR--2023/2948, M3LW-23OR0403025, 2076195, Jun. 2023. doi: 10.2172/2076195.

[22] G. Guarneri, “Weld discontinuities classification using principal component analysis and support vector machine,” presented at the The Brazilian Symposium on Intelligent Automation, Fortaleza, Brazil, Oct. 2013.

[23] Y. Chen, H.-W. Ma, and G.-M. Zhang, “A support vector machine approach for classification of welding defects from ultrasonic signals,” *Nondestructive Testing and Evaluation*, vol. 29, no. 3, pp. 243–254, Jul. 2014, doi: 10.1080/10589759.2014.914210.

[24] K. Virupakshappa and E. Oruklu, “Ultrasonic flaw detection using Support Vector Machine classification,” in *2015 IEEE International Ultrasonics Symposium (IUS)*, Taipei, Taiwan: IEEE, Oct. 2015, pp. 1–4. doi: 10.1109/ULTSYM.2015.0128.

[25] Y. Chen, H.-W. Ma, and M. Dong, “Automatic classification of welding defects from ultrasonic signals using an SVM-based RBF neural network approach,” *Insight - Non-Destructive Testing and Condition Monitoring*, vol. 60, no. 4, pp. 194–199, Apr. 2018, doi: 10.1784/insi.2018.60.4.194.

[26] S. Shevchik *et al.*, “Laser Welding Quality Monitoring via Graph Support Vector Machine with Data Adaptive Kernel,” *undefined*, 2019, Accessed: Sep. 30, 2021. [Online]. Available: https://www.semanticscholar.org/paper/Laser-Welding-Quality-Monitoring-via-Graph-Support-Shevchik-Le-Quang/1d2b3a98eadec5d77574da4cb3a0013fcb5e8512

[27] H. Xiao, D. Chen, J. Xu, and S. Guo, “Defects identification using the improved ultrasonic measurement model and support vector machines,” *NDT & E International*, vol. 111, p. 102223, Apr. 2020, doi: 10.1016/j.ndteint.2020.102223.

[28] S. Legendre, D. Massicotte, J. Goyette, and T. K. Bose, “Neural classification of Lamb wave ultrasonic weld testing signals using wavelet coefficients,” *IEEE Transactions on Instrumentation and Measurement*, vol. 50, no. 3, pp. 672–678, Jun. 2001, doi: 10.1109/19.930439.

[29] Ó. Martín, M. López, and F. Martín, “Artificial neural networks for quality control by ultrasonic testing in resistance spot welding,” *Journal of Materials Processing Technology*, vol. 183, no. 2, pp. 226–233, Mar. 2007, doi: 10.1016/j.jmatprotec.2006.10.011.

[30] S. Sambath, P. Nagaraj, and N. Selvakumar, “Automatic Defect Classification in Ultrasonic NDT Using Artificial Intelligence,” *J Nondestruct Eval*, vol. 30, no. 1, pp. 20–28, Mar. 2011, doi: 10.1007/s10921-010-0086-0.

[31] Z. Chen, G. Huang, C. Lu, and G. Chen, “Automatic Recognition of Weld Defects in TOFD D-Scan Images Based on Faster R-CNN,” *JTE*, vol. 48, no. 2, pp. 811–824, Aug. 2018, doi: 10.1520/JTE20170563.

[32] L. C. Silva, E. F. Simas Filho, M. C. S. Albuquerque, I. C. Silva, and C. T. T. Farias, “Segmented analysis of time-of-flight diffraction ultrasound for flaw detection in welded steel plates using extreme learning machines,” *Ultrasonics*, vol. 102, p. 106057, Mar. 2020, doi: https://doi.org/10.1016/j.ultras.2019.106057.

[33] N. Munir, H.-J. Kim, S.-J. Song, and S.-S. Kang, “Investigation of Deep Neural Network with Drop Out for Ultrasonic Flaw Classification in Weldments,” *Journal of Mechanical Science and Technology*, vol. 32, no. 7, pp. 3073–3080, Jul. 2018, doi: 10.1007/s12206-018-0610-1.

[34] N. Munir, H.-J. Kim, J. Park, S.-J. Song, and S.-S. Kang, “Convolutional neural network for ultrasonic weldment flaw classification in noisy conditions,” *Ultrasonics*, vol. 94, pp. 74–81, Apr. 2019, doi: 10.1016/j.ultras.2018.12.001.

[35] K. Virupakshappa and E. Oruklu, “Multi-Class Classification of Defect Types in Ultrasonic NDT Signals with Convolutional Neural Networks,” in *2019 IEEE International Ultrasonics Symposium (IUS)*, Oct. 2019, pp. 1647–1650. doi: 10.1109/ULTSYM.2019.8926027.

[36] N. Munir, J. Park, H.-J. Kim, S.-J. Song, and S.-S. Kang, “Performance Enhancement of Convolutional Neural Network for Ultrasonic Flaw Classification by Adopting Autoencoder,” *NDT & E International*, vol. 111, p. 102218, Apr. 2020, doi: 10.1016/j.ndteint.2020.102218.

[37] M. Muhlheim, P. Ramuhalli, A. Huning, A. Guler, and A. Saxena, “Status Report on Regulatory Criteria Applicable to the Use of Artificial Intelligence (AI) and Machine Learning (ML),” Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), ORNL/SPR-2023/3072, Sep. 2023.

[38] C. Malone, “Quantitative Assessment of Alkali-Silica Reaction in Small and Large-Scale Concrete Specimens Utilizing Nonlinear Acoustic Techniques,” *Civil and Environmental Engineering Theses, Dissertations, and Student Research*, Aug. 2020.

[39] H. Sun, “Thermal Modulation of Nonlinear Ultrasonic Waves,” Ph.D., The University of Nebraska - Lincoln, United States -- Nebraska. Accessed: Jun. 01, 2023. [Online]. Available: https://www.proquest.com/docview/2444913300/abstract/D84F623EF6674299PQ/1

[40] N. Maeda, “A Method for Reading and Checking Phase Time in Auto-Processing System of Seismic Wave Data,” *Zisin (Journal of the Seismological Society of Japan. 2nd ser.)*, vol. 38, no. 3, pp. 365–379, 1985.

[41] V. Matz, M. Kreidl, and R. Smid, “Classification of ultrasonic signals,” *International Journal of Materials and Product Technology*, vol. 27, no. 3–4, pp. 145–155, Jan. 2006, doi: 10.1504/IJMPT.2006.011267.

[42] K. Virupakshappa and E. Oruklu, “Ultrasonic flaw detection using Support Vector Machine classification,” in *2015 IEEE International Ultrasonics Symposium (IUS)*, Taipei, Taiwan: IEEE, Oct. 2015, pp. 1–4. doi: 10.1109/ULTSYM.2015.0128.

[43] S. A. Shevchik *et al.*, “Laser Welding Quality Monitoring via Graph Support Vector Machine With Data Adaptive Kernel,” *IEEE Access*, vol. 7, pp. 93108–93122, 2019, doi: 10.1109/ACCESS.2019.2927661.

[44] A. Al-Ataby, W. Al-Nuaimy, C. R. Brett, and O. Zahran, “Automatic detection and classification of weld flaws in TOFD data using wavelet transform and support vector machines,” *Insight - Non-Destructive Testing and Condition Monitoring*, vol. 52, no. 11, pp. 597–602, Nov. 2010, doi: 10.1784/insi.2010.52.11.597.

[45] N. A. Akram, D. Isa, R. Rajkumar, and L. H. Lee, “Active incremental Support Vector Machine for oil and gas pipeline defects prediction system using long range ultrasonic transducers,” *Ultrasonics*, vol. 54, no. 6, pp. 1534–1544, Aug. 2014, doi: 10.1016/j.ultras.2014.03.017.

[46] R. Lippmann, “An introduction to computing with neural nets,” *IEEE ASSP Magazine*, vol. 4, no. 2, pp. 4–22, Apr. 1987, doi: 10.1109/MASSP.1987.1165576.

APPENDIX A. CODES PARAMETERS FOR THE MACHINE LEARNING MODELS

APPENDIX A. CODES PARAMETERS FOR THE MACHINE LEARNING MODELS

1. MATLAB codes for the support vector regression (SVR) model

- Model training and testing:

```
clc;clear;
load('Feature_response_ASR_new_E_14.mat')
load('Feature_response_ASR2D_new_E_14.mat')
%load('Feature_response_smallASR.mat')

rng(1);
nex=[14 15 16 371];
FF(:,nex)=[];
Exp_T(nex)=[];
Exp_V(nex)=[];
Dt_int(nex)=[];

nex3=[208 257 275];
FF3(:,nex3)=[];
Exp_T3(nex3)=[];
Exp_V3(nex3)=[];
Dt_int3(nex3)=[];

FF(11:13,:)=FF(11:13,:)*1e4;
FF3(11:13,:)=FF3(11:13,:)*1e4;

%%
%SVM
TestIdx=randsample(length(Exp_T),1);

aa=ones(length(Exp_T),1);aa(TestIdx)=0;
TrnIdx=find(aa==1);
%nFeature=1;
nFeature=[14 3 6 7 10 11];

%nFeature=[14 3 4 7 10 11 ];
%nFeature=2:14;
xtrain=FF(nFeature,:);ytrain=Exp_V(:);
%xtest=FF(:,TestIdx);ytest=Exp_V(TestIdx);
xtest=FF3(nFeature,:);ytest=Exp_V3;

[trainedModel1, validationRMSE1,validationPredictions] = trainRegressionModel_SVM(xtrain,ytrain);
yTrainpred=validationPredictions;
yTestpred= trainedModel1.predictFcn(xtest);

R2_model=R2calculate(ytrain,yTrainpred)
RMSE_model = sqrt(mean((ytrain - yTrainpred).^2))
R2_pred=R2calculate(ytest,yTestpred')
RMSE_pred = sqrt(mean((ytest - yTestpred').^2))
%%
figure(1)
subplot(121)
plot([0 max(Exp_V)],[0 max(Exp_V)],'k-','linewidth',2)
hold on;grid on
plot(Exp_V,yTrainpred,'.-','MarkerSize',8)
ylabel('Predicted expansion (%)')
xlabel('Measured expansion (%)')
subplot(122)
plot([0 max(Exp_V3)],[0 max(Exp_V3)],'k-','linewidth',2)
hold on;grid on
plot(Exp_V3,yTestpred,'.-','MarkerSize',8)
ylabel('Predicted expansion')
xlabel('Measured expansion (%)')


figure(2)
```

```
plot(Dt_int,Exp_V,'linewidth',2)
hold on;grid on
plot(Dt_int,yTrainpred,'.-')
xlabel('Age (days)')
ylabel('Expansion(%)')
legend('Measured expansion','Predicted expansion')
str=['R^2= ' num2str(R2_model,'%.3f') ', RMSE=' num2str(RMSE_model,'%.3f') '%'];
text(mean(Dt_int)/2,mean(Exp_V)*1.5,str)

figure(3)
plot(Dt_int3,Exp_V3,'linewidth',2)
hold on;grid on
plot(Dt_int3,yTestpred,'.-')

%plot(Dt_int3(nn3),yTestpred(nn3),'.b')
xlabel('Age (days)')
ylabel('Expansion(%)')
legend('Measured expansion','Predicted expansion')
str=['R^2= ' num2str(R2_pred,'%.3f') ', RMSE=' num2str(RMSE_pred,'%.3f') '%'];
text(mean(Dt_int3)/2,mean(Exp_V3)*1.5,str)
```

- Codes for the *trainRegressionModel_SVM* function:

```
function [trainedModel, validationRMSE,validationPredictions] = trainRegressionModel_SVM(trainingData, responseData)

% [trainedModel, validationRMSE] = trainRegressionModel(trainingData,
% responseData)
% Returns a trained regression model and its RMSE. This code recreates the
% model trained in Regression Learner app. Use the generated code to
% automate training the same model with new data, or to learn how to
% programmatically train models.
%
%  Input:
%      trainingData: A matrix with the same number of rows and data type as
%       the matrix imported into the app.
%
%      responseData: A vector with the same data type as the vector
%       imported into the app. The length of responseData and the number of
%       columns of trainingData must be equal.
%
%  Output:
%      trainedModel: A struct containing the trained regression model. The
%       struct contains various fields with information about the trained
%       model.
%
%      trainedModel.predictFcn: A function to make predictions on new data.
%
%      validationRMSE: A double containing the RMSE. In the app, the
%       History list displays the RMSE for each model.
%
% Use the code to train the model with new data. To retrain your model,
% call the function from the command line with your original data or new
% data as the input arguments trainingData and responseData.
%
% For example, to retrain a regression model trained with the original data
% set T and response Y, enter:
%   [trainedModel, validationRMSE] = trainRegressionModel(T, Y)
%
% To make predictions with the returned 'trainedModel' on new data T2, use
%   yfit = trainedModel.predictFcn(T2)
%
% T2 must be a matrix containing only the predictor rows used for training.
% For details, enter:
%   trainedModel.HowToPredict

% Auto-generated by MATLAB on 09-Nov-2020 20:57:57
% Extract predictors and response
% This code processes the data into the right shape for training the
```

```
% model.
% Convert input to table
%inputTable = array2table(trainingData', 'VariableNames', {'row_1', 'row_2', 'row_3', 'row_4', 'row_5', 'row_6', 'row_7', 'row_8', 'row_9',
'row_10', 'row_11', 'row_12', 'row_13'});

%predictorNames = {'row_1', 'row_2', 'row_3', 'row_4', 'row_5', 'row_6', 'row_7', 'row_8', 'row_9', 'row_10', 'row_11', 'row_12', 'row_13'};
predictors = trainingData';%inputTable(:, predictorNames);
response = responseData(:);
%isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false, false, false, false];

% Train a regression model
% This code specifies all the model options and trains the model.
regressionSVM = fitrsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 36.07188365440059, ...
    'BoxConstraint', 688.9757300240402, ...
    'Epsilon', 0.0001464922702484744, ...
    'Standardize', true);

% Create the result struct with predict function
%predictorExtractionFcn = @(x) array2table(x', 'VariableNames', predictorNames);
svmPredictFcn = @(x) predict(regressionSVM, x);
trainedModel.predictFcn = @(x) svmPredictFcn(x');

% Add additional fields to the result struct
trainedModel.RegressionSVM = regressionSVM;
trainedModel.About = 'This struct is a trained model exported from Regression Learner R2020a.';
trainedModel.HowToPredict = sprintf('To make predictions on a new predictor row matrix, X, use: \n  yfit = c.predictFcn(X) \nreplacing "c" with
the name of the variable that is this struct, e.g. "trainedModel". \n \nX must contain exactly 13 rows because this model was trained using 13
predictors. \nX must contain only predictor rows in exactly the same order and format as your training \ndata. Do not include the response row or
any rows you did not import into the app. \n \nFor more information, see <a href="matlab:helpview(fullfile(docroot, "stats", "stats.map"),
"appregression_exportmodeltoworkspace")">How to predict using an exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
% Convert input to table
%inputTable = array2table(trainingData', 'VariableNames', {'row_1', 'row_2', 'row_3', 'row_4', 'row_5', 'row_6', 'row_7', 'row_8', 'row_9',
'row_10', 'row_11', 'row_12', 'row_13'});

%predictorNames = {'row_1', 'row_2', 'row_3', 'row_4', 'row_5', 'row_6', 'row_7', 'row_8', 'row_9', 'row_10', 'row_11', 'row_12', 'row_13'};
predictors = trainingData';%inputTable(:, predictorNames);
response = responseData(:);
%isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false, false, false, false];

% Perform cross-validation
KFolds = 5;
cvp = cvpartition(size(response, 1), 'KFold', KFolds);
% Initialize the predictions to the proper sizes
validationPredictions = response;
for fold = 1:KFolds
    trainingPredictors = predictors(cvp.training(fold), :);
    trainingResponse = response(cvp.training(fold), :);
    %foldIsCategoricalPredictor = isCategoricalPredictor;

    % Train a regression model
    % This code specifies all the model options and trains the model.
regressionSVM = fitrsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 36.07188365440059, ...
    'BoxConstraint', 688.9757300240402, ...
    'Epsilon', 0.0001464922702484744, ...
    'Standardize', true);
```

```
    % Create the result struct with predict function
    svmPredictFcn = @(x) predict(regressionSVM, x);
    validationPredictFcn = @(x) svmPredictFcn(x);

    % Add additional fields to the result struct

    % Compute validation predictions
    validationPredictors = predictors(cvp.test(fold), :);
    foldPredictions = validationPredictFcn(validationPredictors);

    % Store predictions in the original order
    validationPredictions(cvp.test(fold), :) = foldPredictions;
end

% Compute validation RMSE
isNotMissing = ~isnan(validationPredictions) & ~isnan(response);
validationRMSE = sqrt(nansum(( validationPredictions - response ).^2) / numel(response(isNotMissing) ));
```

- SVR model parameters trained with alkali–silica reaction (ASR) specimen data:

```
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 36.07188365440059, ...
    'BoxConstraint', 688.9757300240402, ...
    'Epsilon', 0.0001464922702484744, ...
    'Standardize', true
```

- SVR model parameters trained with ASR specimen data without normalization:

```
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 72.10590859602786, ...
    'BoxConstraint', 918.8496105820111, ...
    'Epsilon', 0.0003808007790236055, ...
    'Standardize', true
```

- SVR model parameters trained with alkali–silica reaction sample with 2D confinement (ASR-2D) specimen data:

```
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 15.63424863689961, ...
    'BoxConstraint', 59.35451359432304, ...
    'Epsilon', 0.0002516246160701346, ...
    'Standardize', true
```

- SVR model parameters trained with small ASR specimen data:

```
   'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 3, ...
    'KernelScale', 1, ...
    'BoxConstraint', 0.001086720711998364, ...
    'Epsilon', 0.0007049354106281626, ...
    'Standardize', true
```

- SVR model parameters trained with small ASR-2D specimen data:

```
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 0.495375504610928, ...
    'BoxConstraint', 0.2676484927152948, ...
    'Epsilon', 0.0002538695548278865, ...
    'Standardize', true
```

2. Python codes for the deep neural network (DNN) model training using time-domain signals as input

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.io import savemat
# import seaborn as sns
# Make NumPy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)

import tensorflow as tf
import math
from tensorflow import keras
from tensorflow.keras import layers,models

##Training data
train_features=np.load('SigData2D_processed.npy')
train_features=np.abs(train_features)
train_labels=np.load('Exp_V0_2D.npy')

Val_index=np.arange(0,len(train_labels),5)
Val_features=train_features[Val_index,:]
Val_labels=train_labels[Val_index,:]

train_features1=np.delete(train_features,Val_index,0)
train_labels1=np.delete(train_labels,Val_index,0)

###Testing data
test_features=np.load('SigData_processed.npy')
test_features=np.abs(test_features)
test_labels=np.load('Exp_V0.npy')

nex3=[]
test_features=np.delete(test_features,nex3,0)
test_labels=np.delete(test_labels,nex3,0)

##DNN for regression

model = models.Sequential()
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.4))

model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.4))

# model.add(layers.Dense(64, activation='relu'))
# model.add(layers.Dropout(0.4))

model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(8, activation='relu'))
# model.add(layers.Dense(4, activation='relu'))
# model.add(layers.Dropout(0.3))
model.add(layers.Dense(1))
model.summary

model.compile(loss=tf.keras.losses.MeanSquaredError(),optimizer=tf.keras.optimizers.Adam(0.001))

##time
history = model.fit(
    train_features1,
    train_labels1,
    # validation_split=0.25,
    validation_data=(Val_features,Val_labels),
    verbose=0, epochs=200)
```

```
model.save('ASR2D_signal.h5')

plt.figure(1)
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
loss=history.history['loss']
Val_loss=history.history['val_loss']
# plt.ylim([0, 0.01])
plt.xlabel('Epoch')
plt.ylabel('Error [%]')
plt.legend()
plt.grid(True)
train_predictions = model.predict(train_features).flatten()
test_predictions = model.predict(test_features).flatten()

def plotFitResults(test_labels,test_features,dnn_model):
    test_results= dnn_model.evaluate(test_features, test_labels, verbose=0)
    test_predictions = dnn_model.predict(test_features).flatten()

    plt.figure()
    a = plt.axes(aspect='equal')
    plt.scatter(test_labels, test_predictions,c="red")
    plt.xlabel('Measured expansion (%)')
    plt.ylabel('Predicted expansion (%)')
    lims = [0, max(test_labels)[0]]
    # plt.xlim(lims)
    # plt.ylim(lims)
    plt.grid(True)
    _ = plt.plot(lims, lims,'k-')


    # corr_matrix = np.corrcoef(test_labels[:,0], test_predictions)
    # corr = corr_matrix[0,1]
    # R_sq = corr**2
    y=test_labels[:,0]
    yhat=test_predictions

    ymean=sum(y)/len(y);
    SSE=sum((yhat-y)**2)
    SST=sum((y-ymean)**2)
    R_sq=1-SSE/SST


    MSE = np.square(np.subtract(test_labels[:,0], test_predictions)).mean()
    RMSE = math.sqrt(MSE)

    print([R_sq,RMSE])
    return R_sq, RMSE

[R_sq1,RMSE1]=plotFitResults(train_labels,train_features,model)
[R_sq2,RMSE2]=plotFitResults(test_labels,test_features,model)
```

3. Python codes for the DNN model training using frequency spectra as input

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.io import savemat
# import seaborn as sns

# Make NumPy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)

import tensorflow as tf
import math
from tensorflow import keras
from tensorflow.keras import layers,models
```

```
np.random.seed(1)

##Training data
train_features=np.load('SigData_smallASR2D_FFT_processed.npy')
train_features=np.abs(train_features)
train_labels=np.load('Exp_V_smallASR2D.npy')

# nex=[13, 14, 15, 370]
# train_features=np.delete(train_features,nex,0)
# train_labels=np.delete(train_labels,nex,0)

Esum=np.zeros(len(train_features))
for i in range(1,len(train_features)):
    Esum[i]=np.sum(np.square(train_features[i,:]))
    train_features[i,:]=train_features[i,:]/np.sqrt(Esum[i])*100


Val_index=np.arange(0,len(train_labels),5)
Val_features=train_features[Val_index,:]
Val_labels=train_labels[Val_index,:]

train_features1=np.delete(train_features,Val_index,0)
train_labels1=np.delete(train_labels,Val_index,0)


###Testing data
test_features=np.load('SigData_smallASR_FFT_processed.npy')
test_features=np.abs(test_features)
test_labels=np.load('Exp_V_smallASR.npy')

Esumt=np.zeros(len(test_features))
for i in range(1,len(test_features)):
    Esumt[i]=np.sum(np.square(test_features[i,:]))
    test_features[i,:]=test_features[i,:]/np.sqrt(Esumt[i])*100

##DNN for regression

model = models.Sequential()
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(16, activation='relu'))

model.add(layers.Dense(8, activation='relu'))

model.add(layers.Dense(4, activation='relu'))

model.add(layers.Dense(1))
model.summary

model.compile(optimizer=tf.keras.optimizers.Adam(0.001), loss=tf.keras.losses.MeanSquaredError())

##time
history = model.fit(
    train_features1,
    train_labels1,
    # validation_split=0.25,
    validation_data=(Val_features,Val_labels),
    verbose=0, epochs=150)

model.save("SmallASR2D_signalFFT.h5")

plt.figure(1)
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
loss=history.history['loss']
Val_loss=history.history['val_loss']
# plt.ylim([0, 0.05])
```

```
plt.xlabel('Epoch')
plt.ylabel('Error [%]')
plt.legend()
plt.grid(True)

train_predictions = model.predict(train_features).flatten()
test_predictions = model.predict(test_features).flatten()


def plotFitResults(test_labels,test_features,dnn_model):

    test_results= dnn_model.evaluate(test_features, test_labels, verbose=0)

    test_predictions = dnn_model.predict(test_features).flatten()

    plt.figure()
    a = plt.axes(aspect='equal')
    plt.scatter(test_labels, test_predictions,c="red")
    plt.xlabel('Measured expansion (%)')
    plt.ylabel('Predicted expansion (%)')
    lims = [0, max(test_labels)[0]]
    plt.xlim(lims)
    plt.ylim(lims)
    plt.grid(True)
    _ = plt.plot(lims, lims,'k-')

    y=test_labels[:,0]
    yhat=test_predictions

    ymean=sum(y)/len(y);
    SSE=sum((yhat-y)**2)
    SST=sum((y-ymean)**2)
    R_sq=1-SSE/SST

    MSE = np.square(np.subtract(test_labels[:,0], test_predictions)).mean()
    RMSE = math.sqrt(MSE)

    print([R_sq,RMSE])
    return R_sq, RMSE

[R_sq1,RMSE1]=plotFitResults(train_labels,train_features,model)
[R_sq2,RMSE2]=plotFitResults(test_labels,test_features,model)
```

4. Python codes for the DNN model testing

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.io import savemat
# import seaborn as sns
# Make NumPy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)
import tensorflow as tf
import math
from tensorflow import keras
from tensorflow.keras import layers,models

trained_model='ASR2D_signal.h5'
new_model = tf.keras.models.load_model(trained_model)

##Testing data for ASR-2D
test_features=np.load('SigData_smallASR2D_processed.npy')
test_features=np.abs(test_features)
test_labels=np.load('Exp_V_smallASR2D.npy')

nex3=[] #small ASR
test_features=np.delete(test_features,nex3,0)
test_labels=np.delete(test_labels,nex3,0)
```

```
loss= new_model.evaluate(test_features, test_labels, verbose=2)
test_predictions = new_model.predict(test_features).flatten()

def plotFitResults(test_labels,test_features,dnn_model):
    test_results= dnn_model.evaluate(test_features, test_labels, verbose=0)
    test_predictions = dnn_model.predict(test_features).flatten()
    plt.figure()
    a = plt.axes(aspect='equal')
    plt.scatter(test_labels, test_predictions,c="red")
    plt.xlabel('Measured expansion (%)')
    plt.ylabel('Predicted expansion (%)')
    lims = [0, max(test_labels)[0]]
    # plt.xlim(lims)
    # plt.ylim(lims)
    plt.grid(True)
    _ = plt.plot(lims, lims,'k-')
    # corr_matrix = np.corrcoef(test_labels[:,0], test_predictions)
    # corr = corr_matrix[0,1]
    # R_sq = corr**2
    y=test_labels[:,0]
    yhat=test_predictions
    ymean=sum(y)/len(y);
    SSE=sum((yhat-y)**2)
    SST=sum((y-ymean)**2)
    R_sq=1-SSE/SST
    MSE = np.square(np.subtract(test_labels[:,0], test_predictions)).mean()
    RMSE = math.sqrt(MSE)
    print([R_sq,RMSE])
    return R_sq, RMSE
[R_sq2,RMSE2]=plotFitResults(test_labels,test_features,new_model)

mdic = {"ytest":test_labels,"yTestpred":test_predictions,"nex3":nex3,"loss":loss}
savemat("DNN_ASR2D_smallASR2D.mat", mdic)
```

APPENDIX B. FIGURES FOR RESULTS OF DEEP NEURAL NETWORK MODELS

APPENDIX B. FIGURES FOR RESULTS OF DEEP NEURAL NETWORK MODELS

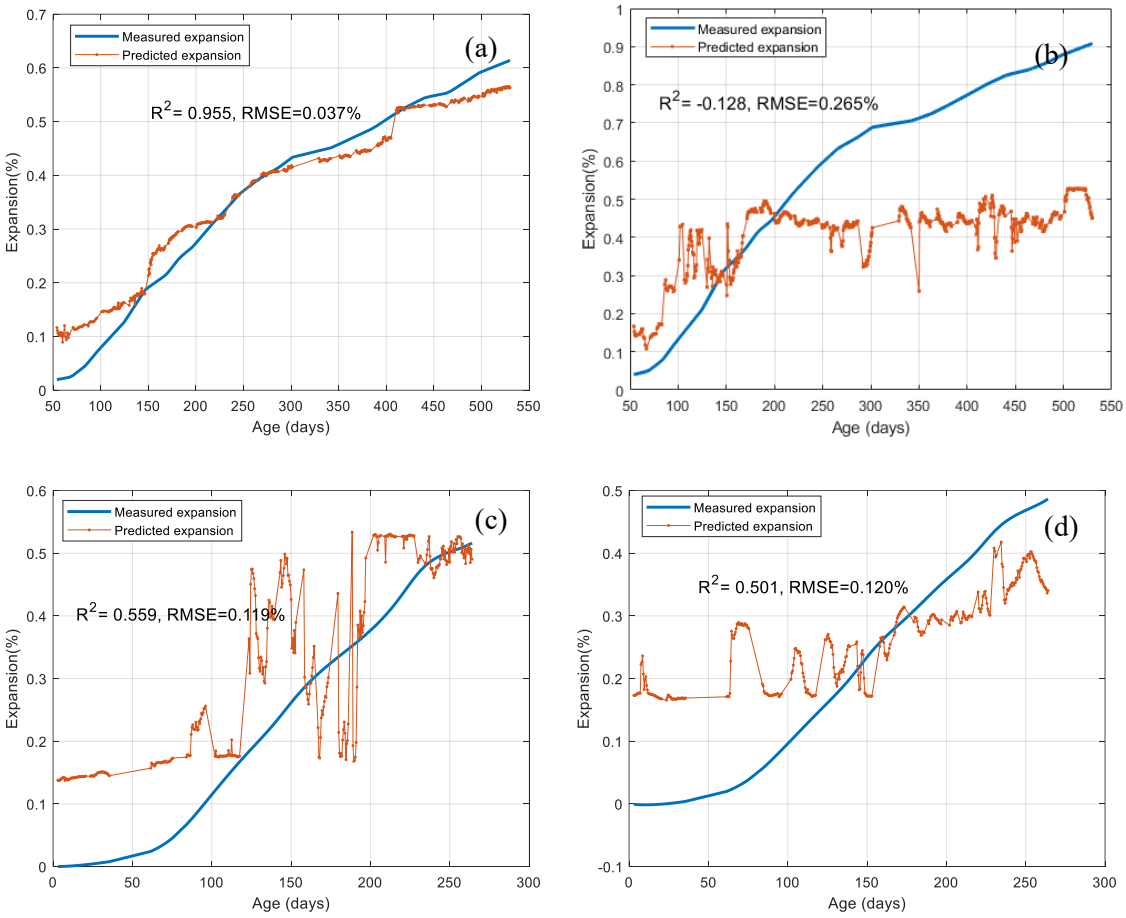

Figure B-1. Deep neural network (DNN) model trained with alkali–silica reaction sample with 2D confinement (ASR-2D) time-domain signals: (a) training results, (b) testing results on the alkali–silica reaction (ASR) data, (c) testing results on the small ASR data, and (d) testing results on the small ASR-2D data.

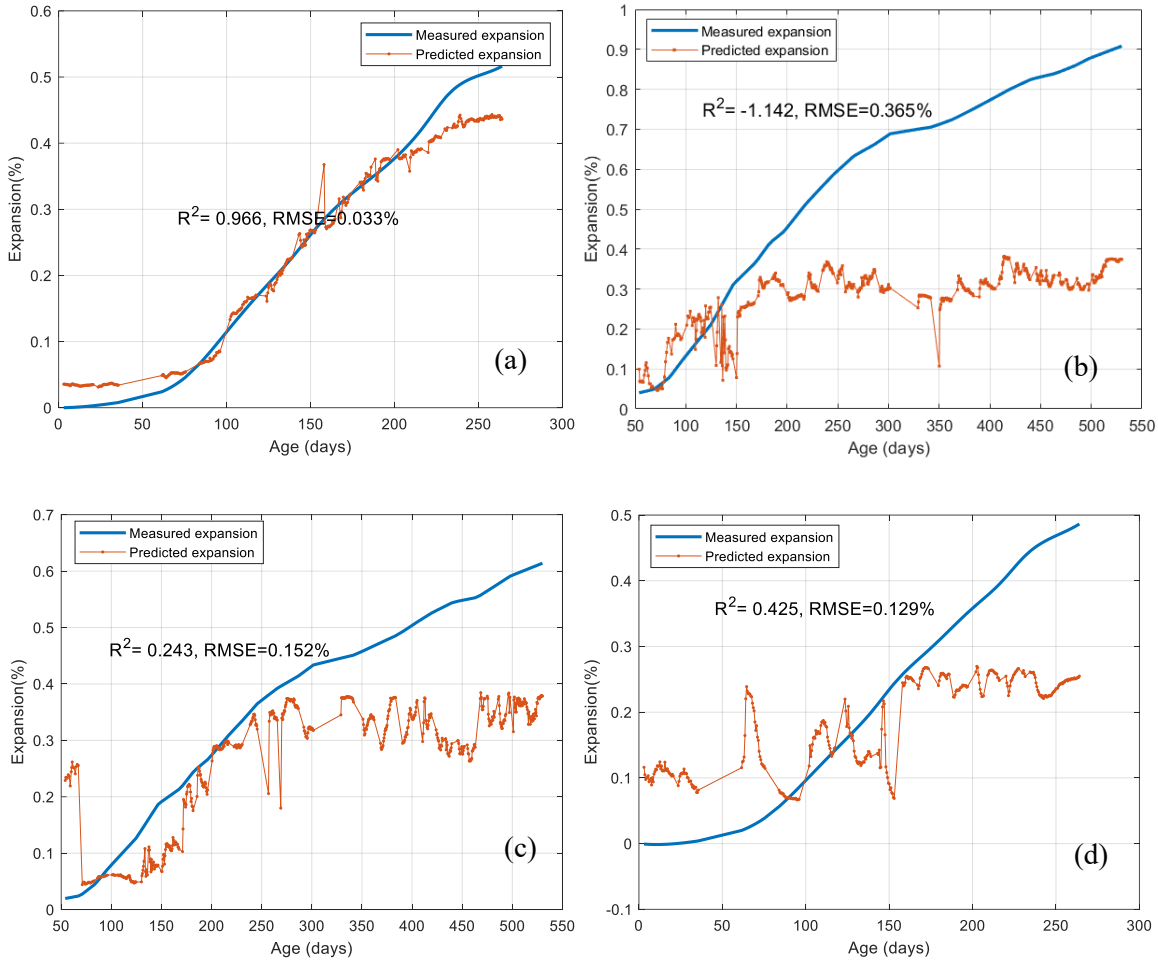

Figure B-2. DNN model trained with small ASR time-domain signals: (a) training results, (b) testing results on the ASR data, (c) testing results on the ASR-2D data, and (d) testing results on the small ASR-2D data.

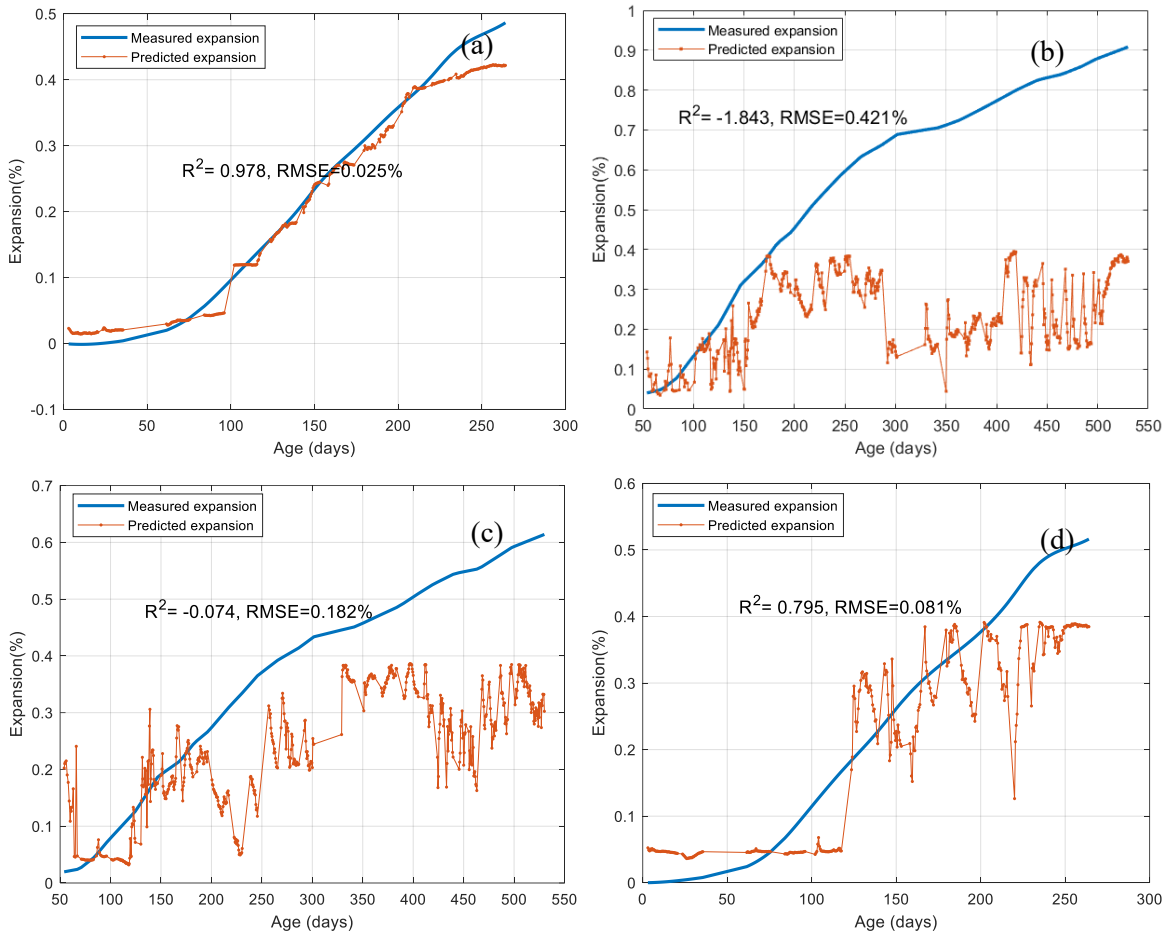

Figure B-3. DNN model trained with small ASR-2D time-domain signals: (a) training results, (b) testing results on the ASR data, (c) testing results on the ASR-2D data, and (d) testing results on the small ASR data.

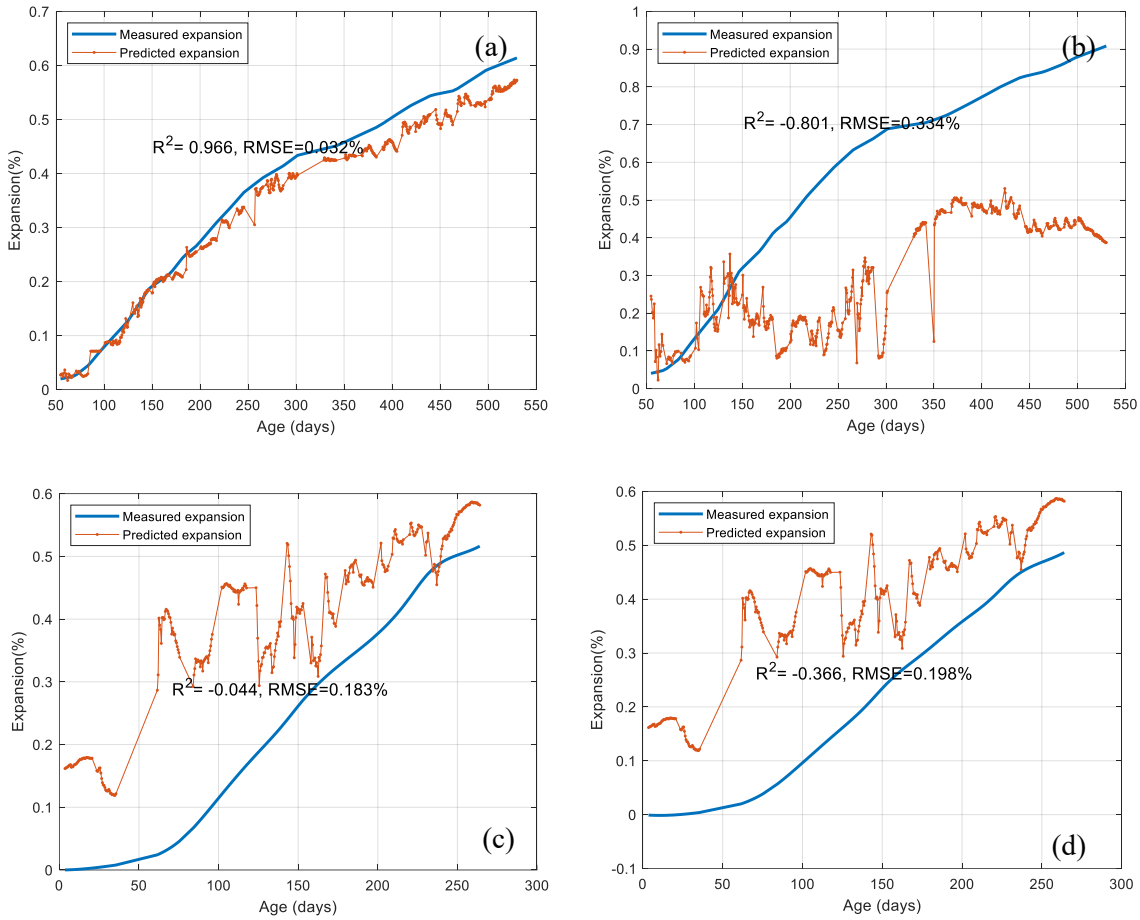

Figure B-4. DNN model trained with ASR-2D frequency spectra: (a) training results, (b) testing results on the ASR data, (c) testing results on the small ASR data, and (d) testing results on the small ASR-2D data.

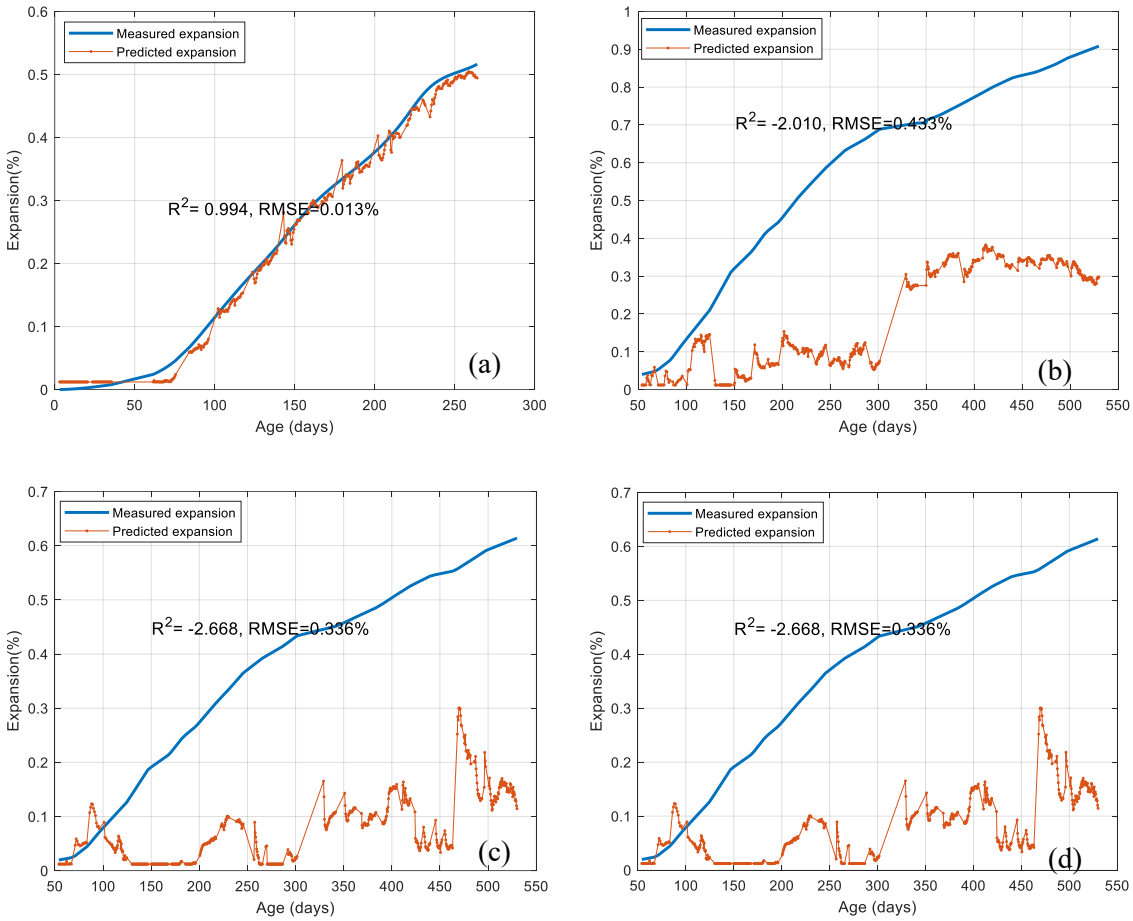

Figure B-5. DNN model trained with small ASR frequency spectra: (a) training results, (b) testing results on the ASR data, (c) testing results on the ASR-2D data, and (d) testing results on the small ASR-2D data.

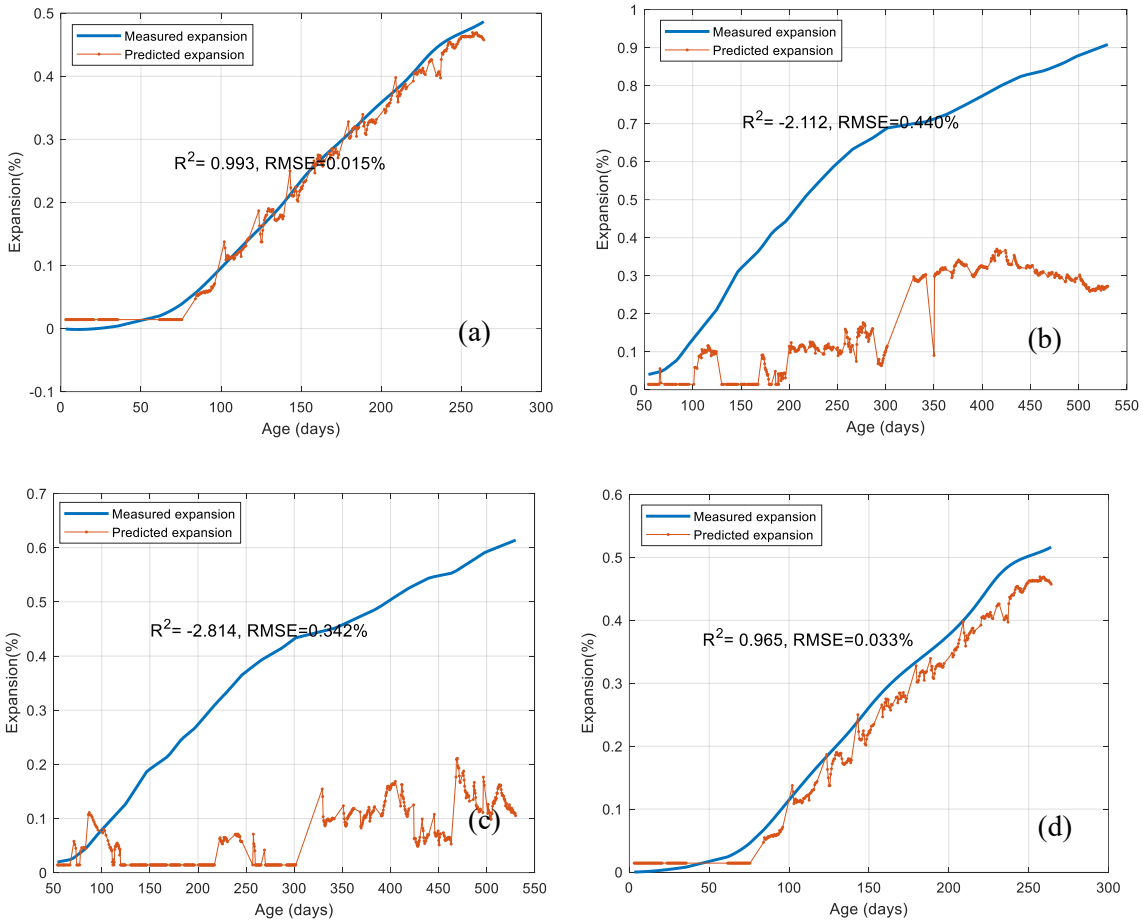

Figure B-6. DNN model trained with small ASR-2D frequency spectra: (a) training results, (b) testing results on the ASR data, (c) testing results on the ASR-2D data, and (d) testing results on the small ASR data.