A METHOD FOR GENERATING SYNTHETIC IMPACT ECHO IMAGES IN STRUCTURAL HEALTH MONITORING

COMPDYN 2025
10th ECCOMAS Thematic Conference on
Computational Methods in Structural Dynamics and Earthquake Engineering
M. Papadrakakis, M. Fragiadakis (eds.)
Rhodes Island, Greece, 15-–18 June 2025

A METHOD FOR GENERATING SYNTHETIC IMPACT ECHO IMAGES IN STRUCTURAL HEALTH MONITORING

Giuseppe Desiderio[1], Leonardo Zunino[1], Maurizio Morgese[2], Valentina Villa[1], Marco Domaneschi[1] and Ali Maher[2]

[1]Politecnico di Torino - DISEG
Turin, Italy
e-mail: {giuseppe.desiderio,leonardo.zunino,valentina.villa,marco.domaneschi}@polito.it

[2] CAIT - Rutgers University
Piscataway, New Jersey
e-mail: maurizio.morgese@rutgers.edu, mmaher@soe.rutgers.edu

Keywords: Structural Health Monitoring, Impact Echo, Deep Learning, Synthetic Data Generation, Defect Detection.

Abstract. *Structural Health Monitoring of civil infrastructures often relies on effective and interpretable assessment techniques such as Impact Echo. Traditional Impact Echo methods require substantial manual effort and specialized equipment, limiting their scalability and ease of deployment. In this paper, a novel deep learning pipeline designed to approximate the Impact Echo function starting from strain gauge sensor measurements collected from bridge deck is proposed. By leveraging high-fidelity data provided by The BEAST (Bridge Evaluation and Accelerated Structural Testing) facility, our pipeline transforms these measurements into interpretable Impact Echo images, producing output patches of size* 128×128 *pixels that effectively visualize structural integrity. Addressing the prevalent challenge of limited datasets in Structural Health Monitoring, a custom algorithm for synthetic image generation is introduced to augment the available data, thereby improving the robustness and performance of our deep learning model. The experimental evaluation demonstrates that our pipeline accurately approximates the impact echo signals, providing an automated and interpretable approach for structural health monitoring.*

1 INTRODUCTION

Structural health monitoring (SHM) plays a crucial role in ensuring the integrity and safety of civil infrastructure, including bridges, buildings, and critical transportation arteries. Effective SHM strategies are essential for timely damage detection, optimized maintenance scheduling, and minimizing resource expenditure [1]. In the context of SHM, Non-Destructive Testing (NDT) is commonly used to detect defects without causing damage to the structural integrity of the inspected component. Among the various NDT techniques employed, the Impact Echo (IE) method is a typical choice for its robustness and interpretability in detecting subsurface defects in concrete structures [4]. In fact, IE is a well-established non-destructive testing widely used in civil engineering for the evaluation of concrete and other construction materials. The fundamental principle underlying this method is the propagation and reflection of stress waves. The presence of defects has been observed to cause alterations in reflection patterns and introduces additional frequency peaks [4]. However, as most traditional NDTs, they often present practical limitations due to their reliance on specialized equipment, intensive manual operation, the need for expert interpretation of acquired signals, and can be time-consuming, especially for large structures. These factors can hinder the scalability and widespread deployment of IE for a comprehensive assessment of the infrastructure.

Recent advances in Deep Learning (DL) offer promising avenues for automating and enhancing SHM practices. In particular, DL techniques approaches for vibration-based SHM span a diverse range of techniques, from classic methods [2], [3], to recent deep learning solutions such as Convolutional Neural Networks (CNNs) [5], Autoencoders [6], [7], and more recently even Transformer-based architectures [8]. This paper introduces a novel DL pipeline designed to approximate the Impact Echo *condition maps* - summarized in 2D images, directly from strain gauge sensor measurements. This approach aims to fulfill the goal of transforming complex sensor data into readily interpretable visual representations, facilitating faster and more objective structural condition assessments. Using high-fidelity data acquired from experiments conducted at The BEAST (Bridge Evaluation and Accelerated Structural Testing) facility, our pipeline effectively transforms strain measurements taken from bridge decks into IE images, generating output patches of size 128×128 pixels that visually represent structural integrity.

Another important challenge in advancing DL for SHM is the scarcity of large, labeled datasets representative of diverse structural conditions and defect scenarios. To address this limitation and enhance the robustness of the proposed model, this article introduces a custom algorithm for synthetic Impact Echo image generation. This synthetic data augmentation strategy serves to enrich the available experimental data, thereby improving the generalization capability and performance of our deep learning model. The experimental evaluation of our pipeline, using both real-world data from the BEAST facility and synthetic data generated from them, demonstrates its ability to accurately approximate Impact Echo signals and provides an efficient and scalable approach for structural health monitoring.

In summary, the contributions of this paper are:

- The implementation of a DL pipeline that transforms strain measurements into patches of size 128 × 128 pixels that visually represent the structural integrity of bridge decks.
- The definition of a custom data generation algorithm to facilitate the creation of synthetic images, thus enhancing the available experimental data.

While Impact Echo offers significant advantages in terms of its non-destructive nature and ability to probe into the material, traditional IE analysis often requires specialized equipment,

skilled operators to perform tests and interpret the signals, and can be time-consuming, especially for large structures. The interpretation of frequency spectrum can also be subjective and influenced by noise and signal complexity. Therefore, the development of automated and robust methods for IE data analysis, such as the deep learning pipeline proposed in this paper, has the potential to significantly enhance the efficiency, objectivity, and scalability of structural health monitoring applications.

2 Related Works

In the article [9], *Huang et al.* demonstrates state-of-the-art results across a range of audio and speech classification tasks, showcasing the effectiveness of their self-supervised pre-training methodology and spectrogram-based input representation. Their work provides a strong precedent for utilizing spectrograms and an AutoEncoder (AE) architecture for learning meaningful representations from time-series data. This paper has inspired the model design and the insights of analyzing straing gauge measurements into the domain frequency rather than time domain.

On the other hand, *Najafi et al.* [10] addresses the challenge of limited datasets in SHM. A key aspect of their research is the generation of synthetic visible deterioration maps using a heuristic-based algorithm, guided by IE condition maps. Furthermore, their study utilizes data from the BEAST facility, the same experimental data of our research, albeit for different purposes and sensor modalities. This paper serves as inspiration for the development of our custom algorithm for synthetic image generation.

3 CASE STUDY

3.1 The BEAST facility

The Bridge Evaluation and Accelerated Structural Testing (BEAST) facility, developed by the Center for Advanced Infrastructure and Transportation (CAIT) at Rutgers University, is a forefront laboratory dedicated to infrastructure research. It enables simulation of realistic environmental and traffic-induced stressors on full-scale bridge components under precisely controlled laboratory conditions. The unique capability of the BEAST lies in its ability to significantly accelerate structural deterioration processes, compressing years of environmental and operational stress into mere months.

A fundamental component of the BEAST facility is its extensive sensor network, including strain gauges, accelerometers, and temperature sensors, which produce comprehensive multi-modal datasets reflecting the structural response under diverse conditions. The facility's data richness makes it especially valuable for developing and validating SHM techniques, notably in areas such as damage detection, condition assessment, and predictive maintenance.

For this study, a subset of the BEAST's extensive sensor network is utilized, specifically the deck-embedded strain gauges (Geokon Model 4911). A total of 40 strain gages are strategically placed along the girders of the bridge's concrete deck. These sensors measure longitudinal and transversal train within embedded reinforcing bars.

Figure 1 presents a schematic view of the instrumented deck specimen, illustrating sensor positions along its longitudinal and transversal cross-sections.

4 METHODOLOGY

In this section, is explained step by step the algorithm for the generation of synthetic images, the architecture and design choices behind the implementation of the DL pipeline.

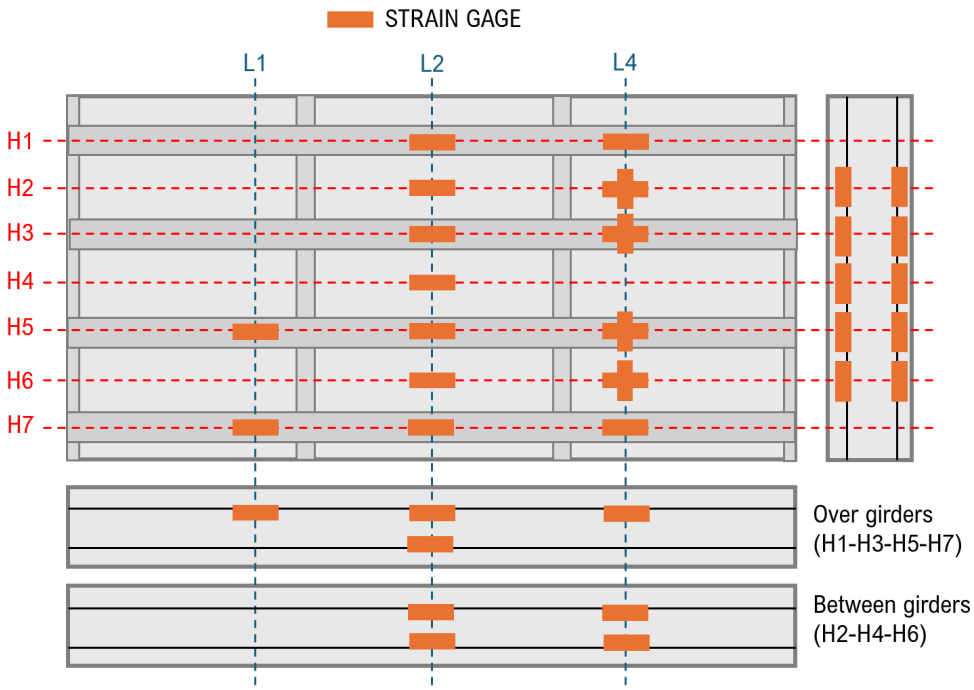


Figure 1: Top and lateral view of the Beast strain gages placement.

4.1 Synthetic Data Generation

One of the main challenges faced is the scarcity of available data. The dataset contains only 11 impact echo images produced at the end of each test, that significantly restrict the application of deep learning models, with so few samples, there is a high risk of *overfitting*, e.g. memorizing the training set instead of learning a generalizable function. To address this issue, this works proposes a custom algorithm for synthetic image generation starting from these 11 images. The main idea is starting from a real-image, suppose the impact echo at time T_i, that has a particular deck condition. The desired outcome is an image representing the degradation backward in time, supppose T_{i-1}. This fake image would then illustrate the deck's condition before the *real* impact echo measurement is conducted. This choice is justified because each test is actually divided in several subtests, and the impact echo is perfomed at the end of each *set* of subtests. Consequently, when the pairs (time-series; image patch) were constructed, the criterion was to prioritize tests that are closest to the date of the impact echo, as their time-series better reflect the bridge's dynamics. This choice results in the subtests "in the middle" lacking a proper target. Thus, it is necessary to generate synthetic images for these tests to be associated with their time series. The algorithm revolves around the technique of image inpainting, a digital image processing technique used to fill in missing or corrupted regions of an image [11]. In the following, the algorithm is explained step by step with details in each phase, all the functions used comes from *OpenCV* library [12].

1. **Initialization:** At the beginning are selected the *original image* Impact Echo at time T_i and a background image, both images have the same size 1488×682. Since in the impact echo, the blue represents non-damaged spots, the background is an RGB image with color $[255, 0, 0]$. In the following, the background will be referred to as I_1, and the impact echo image as I_2.

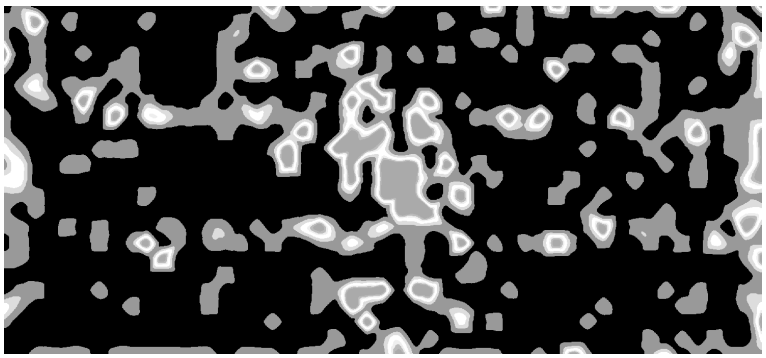


Figure 2: Difference Map between impact echo and background.

2. **Generate the Damage Mask:** This phase is divided into 2 steps: in the first one a difference map is computed, basically is calculated the absolute difference between I_2 and I_1 over each color channel and take the average to get a scalar $\delta(x, y)$ - the average is computed over the channel dimension, resulting in a matrix of shape (H, W) that contains the delta values. The result of this phase is shown in Figure 2. In the second step, once the background is removed from the image, it is necessary to threshold the difference map to isolate damaged areas. This is implemented in a multi-level fashion, indicating that different ranges have been selected to extract damage at different levels. The following list contains the intervals ($[min, max]$) for each level: $[170; 188]$ = Red (serious damage), $[238; 254]$ = Shades around red, $[187; 238]$ = Shades around yellow, $[221; 255]$ = Yellow (poor), $[153; 255]$ = Green (fair). The mathematical formulation is simple if $min \leq \delta(x, y) \leq max$, falls in the interval the pixel in the mask will be set to 1, otherwise it is set to 0, in this way a binary mask is created. Note that more than a single run of this algorithm is performed for each image, since each run produces only a single damage mask, according to the interval that has been chosen; for subsequent runs the resulting image should be given as input for the algorithm as I_2 (Impact Echo). The procedure chosen is to start from the most serious damage (Red) and progress through to the lowest level, the green one; running the algorithm in this order preserves better the visual features during inpaiting. Also notice that for some images there are no "red zones" present, in these cases the process starts from the highest level damage - Yellow or Green. Before passing to the next step the mask obtained is cleaned up with a morphological operator *Open*, this removes small spurious specks and fills tiny holes inside damage regions.

3. **Simulate Partial Growth:** The basic idea that is implement here is that "Damage often expands geometrically over time", so is reasonable to partially shrink the newly formed damage region to emulate a midpoint in time - recall that the starting point is an image at time T_i and we want to produce as output an image at time T_j with $j < i$. The first step in this phase is to find the external contours corresponding to a damaged region. This is possible thanks to the procedure provided by *OpenCV: findContours*, that extracts them from the binary mask. Then each damaged region is shrinked in a random way, meaning that a new kernel is created using *OpenCV: Erode* to simulate partial growth, the size of the kernel is fixed at $(5, 5)$; while the number of iterations is a random choice between 1 and 3, both these parameters control how much the damage region shrinks. An example of a damage mask before and after the "erosion" is presented in Figure 3. During this phase

the damaged regions are sorted according to their area, it is important to calculate the area of each damaged region since a method to "exclude" some damaged regions between T_j and T_i has been implemented; this choice can be justified because some spots that are present in T_i can naturally come up as "new damage". Anyway, the process of cutting off damaged regions according to the area is done only at the end, when the algorithm processes the last interval, the green one. Observation indicates that applying this strategy at each level cancels out too much damage. A threshold value is set according to the $35th$ percentile of the areas, and the probability of retaining damage is set to 30%. During the processing of the green interval, each damaged spot below the threshold (*tiny spot*) has a 30% probability of being retained within the mask.

(a) (b)

Figure 3: (a) Damage Mask after completion of phase 2; (b) Damage Mask after the application of Erode kernels.

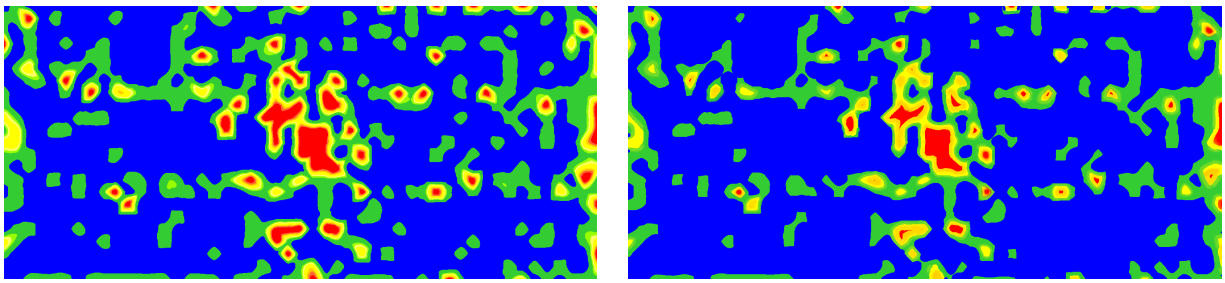
4. **Apply Inpainting:** At this point, the algorithm generates two masks, one containing damage at time T_i and the second one containing the presumably damaged area at time T_j, with $j < i$. Now, the main idea of inpainting is to remove the damage from I_2 (our original image) to yield I_{2Clean}, which will later be used to blend partial damage. Since, the objective in this stage is to completely remove the damage from I_2, the original mask, resulting from step 2, is used here in place of its eroded version. At the end, the algorithm produces I_{2Clean} which is I_2 with damage visually removed (inpainted). As algorithm for the inpaiting *OpenCV: Inpaint_Telea* [13] has been used.

   Then the second step of the inpainting, called blending, is, obviously, the insertion of the "partially grown" damage - eroded mask - into I_{2Clean}. A simple approach, at pixel level, consists of a linear combination between the two images, I_2 and I_{2Clean}, mathematically speaking

$$I_{new}(x,y) = \begin{cases} (1-\alpha)\cdot I_{2Clean}(x,y) + \alpha \cdot I_2(x,y) & \text{if } M_{eroded}(x,y) = 1 \\ I_{2Clean}(x,y) & \text{otherwise} \end{cases} \tag{1}$$

   α controls how strongly you mix the final damage texture. The choice is to set $\alpha = 1$, inside that eroded shape to get the fully damaged pixel from I_2. All the values in between is a different blend between the damage texture and the clean background. An important observation is that the eroded mask is smaller than the full damage region, so only a partial is re-introduced. Outside this partial region, the inpainted background remains.

5. **Post-Processing:** The very last step is application of *OpenCV: bilateralFilter* to further smooth color transitions.



(a) (b)

Figure 4: (a) Original Image at timestamp T_i; (b) Fake Image at timestamp T_j , with $j < i$.

In Figure 4 is shown an example of a real vs. fake image generated with this procedure. It is important to note that to obatin this final result the process should be iterated more than one time. This final result comes up after several iteration of the five steps defined before.

Following the procedure applied to all 11 tests, 7 fake images are generated.

4.2 Heterogeneity in Input vs Output

This study focuses on the application of a deep learning framework able to reproduce Impact Echo images starting from strain gauge measurements. The first thing to note is that there is heterogeneity between inputs and outputs, indeed, strain gauge measurements is a time series data, while the IE data is an RGB image. Thus, it is necessary to design an architecture that converts sensor time-series directly into a 2D matrix, which is not straightforward. Drawing inspiration from [9], the model is not directly applied in the time-domain. Instead, **spectrograms** associated with each time-series are utilized. In this way, the problem translates into a form of *image-to-image* translation, as a spectrogram is treated as a 2D grid, analogous to an image. Therefore, the architectural choice is an AutoEncoder-like structure that translates spectrograms into impact echo representations. Another significant design choice is that the target image is not the entire impact echo image presented previously. Rather, the image has been partitioned into patches of size 128×128. Each patch is centered at the sensor location; border sensors have a shifted center to match the patch size. Therefore, the position of the sensors within the image is mapped, and the associated patch is extracted. This choice implies the assumption of *locality*, meaning that each sensor describes only (or nearly only) the surrounding area, with long-range effects excluded. In summary, it is assumed that most of the information collected by a sensor pertains to the local area surrounding it.

4.3 Pre-Processing

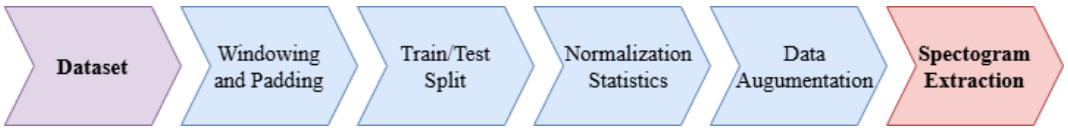


Figure 5: Preprocessing Data Pipeline.

The complete data processing pipeline used in this work is shown in Figure 5. This section outlines the preprocessing applied before feeding the autoencoder with data. The first preprocessing step is **windowing**, i.e., the division of the time series data into time windows, which

the algorithm will process. The data set is composed of time series of different lengths, because different tests do not have the same duration, the dimension of each window is set to comprise 480 data points, that translates into $24h$ of monitoring, with 40% of overlaps between two consecutive windows. Since some time-series could not match the exact division **zero-padding** has been implemented, if needed. At the end, the data set counts 11466 windows, whose 3354 are associated to the fake data. Then the dataset is built according to the common $80-20\%$ percentages for training and test split. Once the training set is defined, the normalization statistics is calculated **only** on the training set, these will be used to normalize samples during training, two normalization strategies have been implemented *zscore* and *minmax*, during experiments the former has beeen found more suitable. It's important to notice that such statistics are calculated on the spectrograms, since the input of our neural network will be spectrograms.

To further increase data diversity a **data augmentation** pipeline has been defined. All the sequence of operations listed below is applied, each one with a certain probability of being realized. In the following the list of transformation considered along with the associated probabilities.

- *Jittering:* Add Gaussian noise to the time series, $\sigma = 0.07$, with a probability $p = 0.9$;
- *Scaling:* Scale the time series by a random factor in the interval $(0.9, 1.1)$, with a probability $p = 0.85$;
- *Time Shifting:* Shift the time series by a random number of steps $max = 15$, with a probability $p = 0.8$;
- *Time Warping:* Warp the time axis using a smooth random curve, simulating variations in the speed or timing of events. Probability of realization $p = 0.6$;
- *Magnitude Warping:* Multiply the time series by a smooth random curve, modifying the amplitude of the signal over time, simulating changes in the intensity. Probability of realization $p = 0.4$.

The last step is the derivation of the spectrogram, the function from the *scipy.signal* library *spectrogram* has been used with the following hyperparameters: frame length of 256, meaning that the FFT is calculated over 256 points, and a step length of 16, the strain gauge sensors have a sample rate of 20 Hz. With this configuration the framework produces spectrograms of size $(129, 15)$, this will be the input for the neural network.

4.4 Model Architecture

Recall that the objective is a supervised image-to-image translation, where the inputs are spectrograms derived from time series, while the outputs are image patches of size 128×128. As an architectural choice, an AutoEncoder-like (AE) [14] architecture was utilized. This is not a "canonical" autoencoder because the resolution of the target, 128×128, does not coincide with the resolution of the input, 129×15, while in a typical autoencoder problem the goal is to reconstruct the input. The architecture is presented in Figure 6. The encoder is composed by a sequence of convolutional blocks: *[Conv, Batch_Norm,ReLU]*, the role of the encoder is commonly known as **feature extractor**. On the other hand, the decoder is composed of blocks which use the so-called transposed convolution in place of normal convolution to upsample the feature maps at each step. In between is placed a fully connected component that acts as projection layer; this is important since the output of the encoder must match the input of the

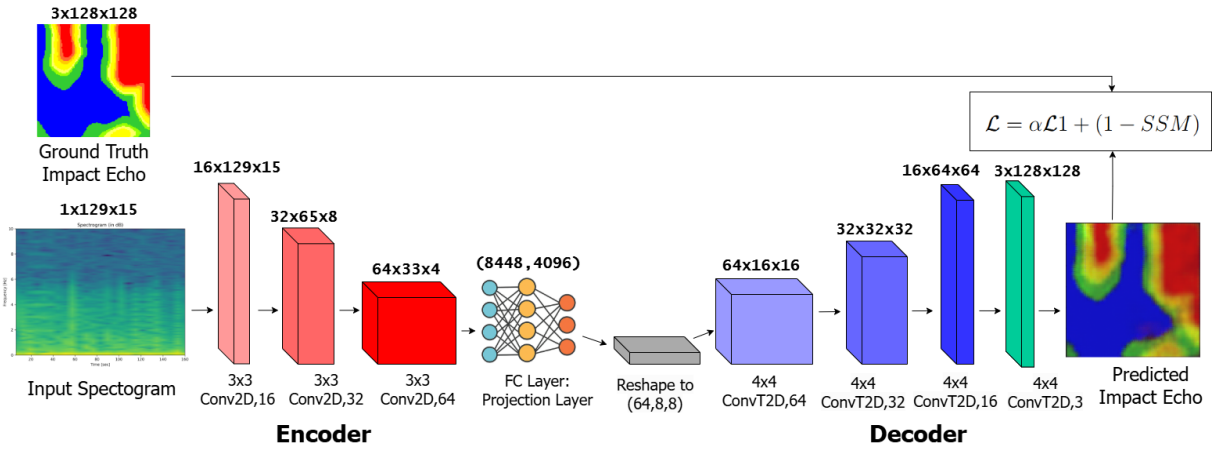


Figure 6: Architecture.

decoder. Typically, if one apply the same operation symmetrically between encoder and decoder there is no need for this projection layer, but recall that the spectrogram input has a different resolution compared to the target, so this layer is necessary here.

Loss Function: To train the model the loss function employed is shown in Eq. 2

$$\mathcal{L} = \alpha\mathcal{L}1 + (1 - SSIM) \tag{2}$$

In Eq. 2, the loss is the contribution of two terms; the first one calculates the $L1$ distance (Mean Absolute Error) between each pixel of the target and predicted image, while the second term is known as Structural Similarity Index Measure (SSIM) [15]. Unlike traditional pixel-wise metrics (e.g., Mean Absolute Error), which only quantify raw intensity differences, SSIM evaluates images based on key perceptual factors. This ensures that generated images preserve perceptual and structural aspects (e.g., edges, textures), producing more visually realistic results. Using the combination of $L1$ and SSIM as in Eq. 2 the goal is to try to balance perceptual quality vs. pixel accuracy.

5 Experiments and Results

This chapter will present the experimental methodology employed to investigate how the AutoEncoder (AE) works effectively in the context of image generation. Recall that the primary goal of this study is the incorporation of an algorithm for the generation of fake images. The resulting images should be consistent with the original ones, aiming at increasing the generalization ability of the neural network. In addition, the section will show the results obtained from different experiments and discussions of their implications in relation to the research questions posed.

5.1 Experiments

Hardware: All experiments were carried out on a workstation equipped with an Intel Core i7-9750H CPU @ 2.60GHz, 32 GB of RAM and a single NVIDIA GTX 1660Ti with 6GB of VRAM.

Experimental Settings: The architecture used in the experimets is shown in Figure 6. The model has been trained for 400 epochs with *AdamW* optimizer [16] with the following hyper-parameters fixed *betas*$= (0.9, 0.95)$ and *weight_decay*$= 1e-4$, initial learning rate of $1e-4$, then we employed as scheduler *CosineAnnealing* [17], batch size of 64 for training and validation sets. The model is trained from scratch, no pre-trained weights.

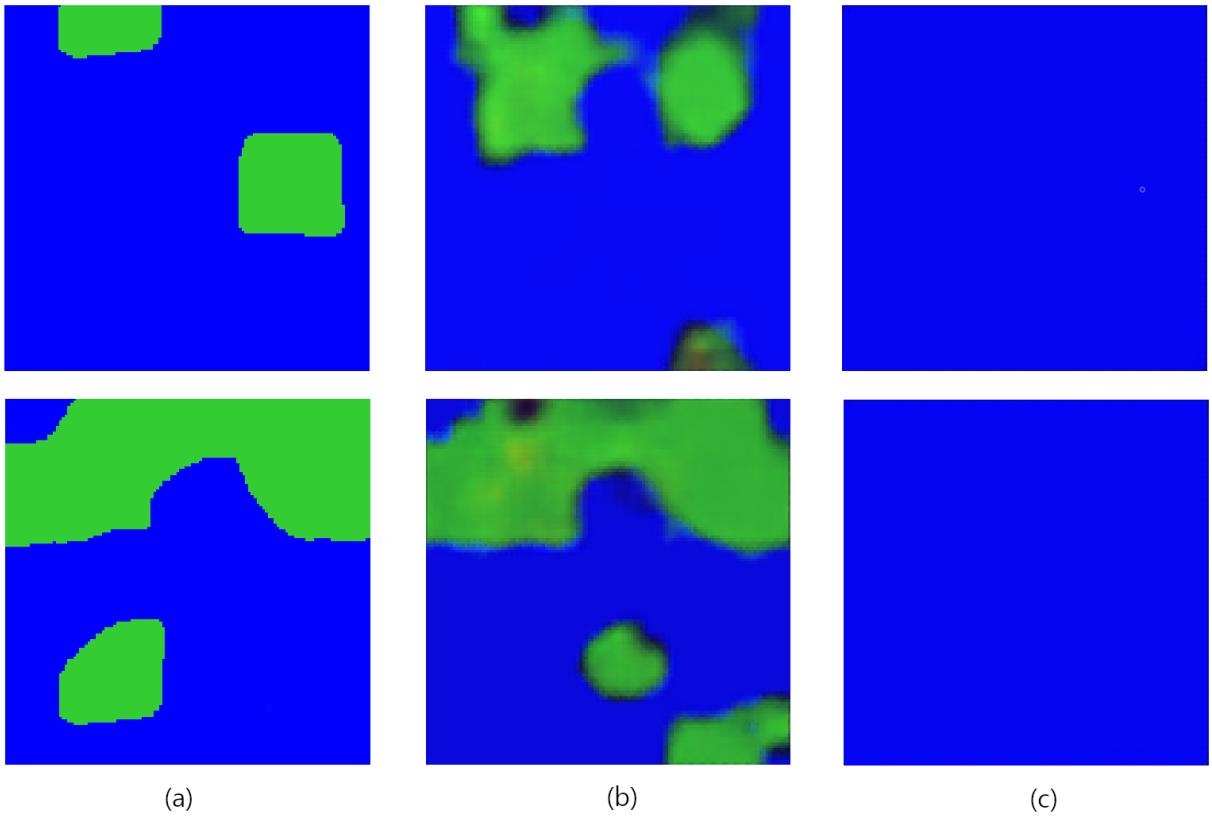


Figure 7: (a) Ground Truth Image, (b) Spectrogram AE predictions, (c) Wavelet AE predictions

5.2 Results

This section shows the results relative to the experiments made in order to assess the quality of the architecture in the context of image generation. The evaluation of image generation is mainly a matter of visual inspection of the final results compared with the ground truth. In this work, a comparison between two architectures, two AutoEncoders, has been made. The architectures have the same size in terms of convolutional layers for encoder-decoder, but the ”alternative” architecture takes as input *wavelet* coefficients rather than spectrograms. This comparison is also made to justify the importance of preprocessing time series with *spectrogram*, indeed wavelet is also a common preprocessing function that is widely applied. Figures 7 and 8 show the results of image generation, in the first column is shown the ground truth image, the target image, while in the last column the prediction made by the autoencoder using wavelet transformation; in the middle there is the prediction from spetrogram analysis. As is evident from the results in Figure 7, using the wavelet pre-processing completely miss the prediction of the patch, leading to an image composed of only background. In contrast, although not perfectly, the predictions from the spectrogram correctly identify different spots in the image that

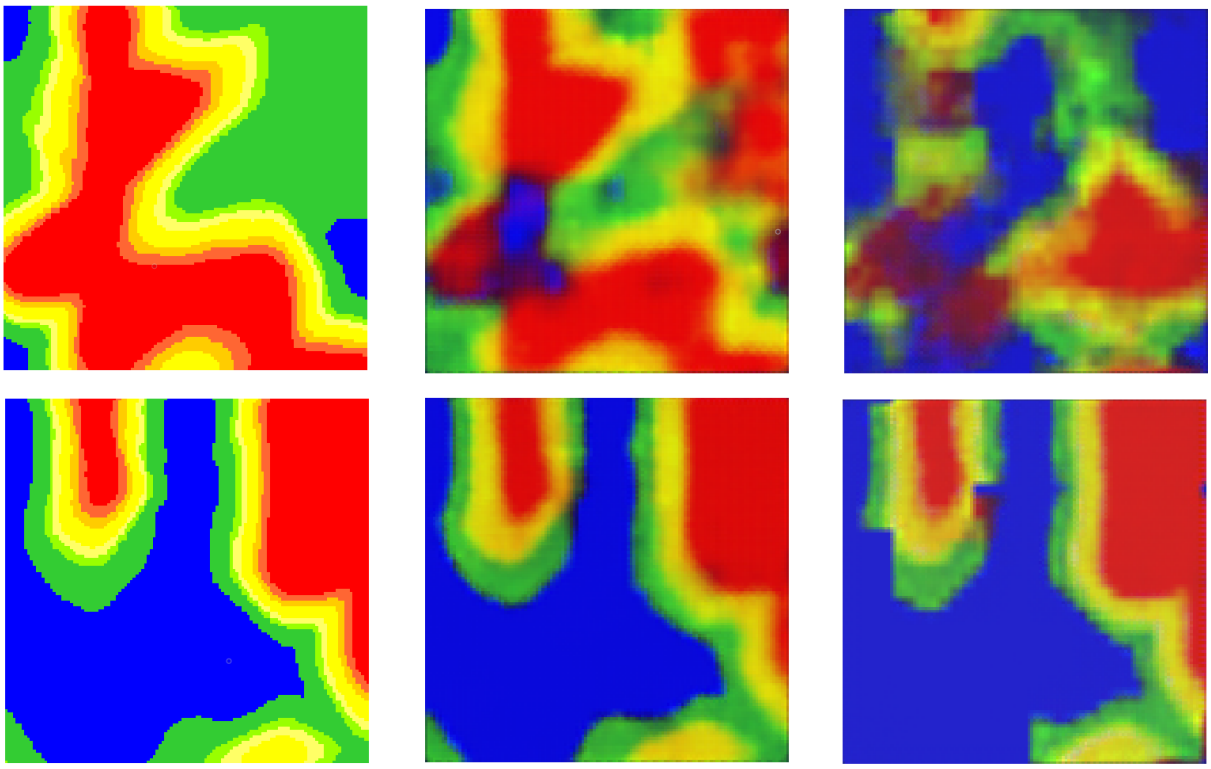


Figure 8: a) Ground Truth Image, (b) Spectrogram AE predictions, (c) Wavelet AE predictions

represent different deck conditions. Figure 8 displays two patches that represent serious damage in that portion of the deck. Observing the patch at the bottom, both architectures correctly identify the shape and the colors, which represents the severity of the damage. However, it can be seen that the patch prediction from spectrogram AE is more precise in terms of details, indeed the bottom left corner is left blue in the wavelet prediction. Moreover, the boundaries are smoother and really close compared to the ground truth. A similar reasoning can be made for the patch located at the top of Figure 8. In this case, the wavelet prediction resembles just random noise without any structure; while the prediction from the spectrogram AE correctly determines the shape of the yellow boundary and, even though not completely, the red zone inside.

Anyway, the model is not exempt from bad predictions. Figure 9 shows an example in which the AE completely fails in predicting the patch. The motivations can be several, the time series is devoid of information; indeed a problem of the windowing approach is that the time series is divided into chunks, thereby compromising the comprehensive perspective. In general, the framework, probably, still miss some important details, but it's important to notice that this is just a preliminary analysis aiming at evaluating the feasibility of transforming time series data obtained from strain gauge into image patches, impact echo representing deck integrity.

6 Conclusions

In conclusion, this article has explored the field of image generation in the context of structural health monitoring. Specifically, the article fulfills the goal of generating consistent impact echo images. Such images can be used alongside real image in order to train a deep learning

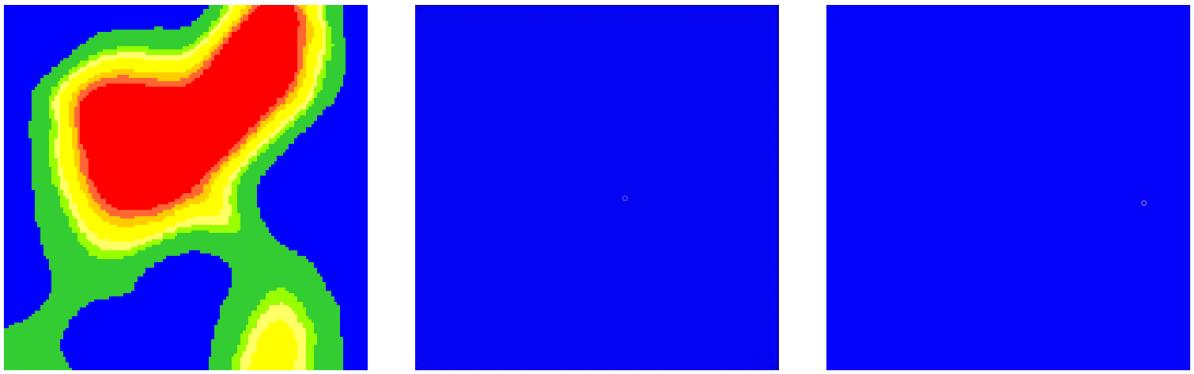


Figure 9: a) Ground Truth Image, (b) Spectrogram AE predictions, (c) Wavelet AE predictions

framework that takes as input strain gauge measurements and generates impact echo images. The natural development of this work is to reconstruct the entire image. An idea could be the introduction of a second network that works on top of the autoencoder, this network should learn how to unify the patches to generate a high-resolution IE image. Alternatively, there is an "all-in-one" approach: training a single "large" network that takes all sensor data as input and directly produces a single 2D image at desired resolution - GPU memory can become an obstacle here.

Future Works: Anyway, there is still a lot of work to do, indeed it has been demonstrated that the model does not cover all the cases well. In particular, it is important to explore new architectures such as the Generative Adversarial Network (GAN) [18], which has been demonstrated impressive results in image generation, but are more difficult to train and stabilize. Also, another improvement is to extend the window size in order to reproduce more informative spectrograms, the choice made here is a trade-off between data availability and computational costs.

Acknowledgments

The authors gratefully acknowledge the Center for Advanced Infrastructure and Transportation (CAIT) at Rutgers University for providing access to the BEAST datasets used in this study.

REFERENCES

[1] Malekloo A, Ozer E, AlHamaydeh M, Girolami M. *Machine learning and structural health monitoring overview with emerging technology and high-dimensional data source highlights. Structural Health Monitoring.* 2021;21(4):1906-1955. doi:10.1177/14759217211036880

[2] Alessio Burrello, Giovanni Zara, Luca Benini, Davide Brunelli, Enrico Macii, Massimo Poncino, Daniele Jahier Pagliari, *Traffic Load Estimation from Structural Health Monitoring sensors using supervised learning,* Sustainable Computing: Informatics and Systems, https://doi.org/10.1016/j.suscom.2022.100704.

[3] Amaral, Rafaelle Cury, Alexandre Barbosa, Flávio. (2019). *An SHM approach using machine learning and statistical indicators extracted from raw dynamic measurements.* Latin American Journal of Solids and Structures. 10.1590/1679-78254942.

[4] Carino, Nicholas. (2001). *The Impact-Echo Method: An Overview*. 10.1061/40558(2001)15.

[5] Kim, Soon-Young Mukhiddinov, Mukhriddin. (2023). *Data Anomaly Detection for Structural Health Monitoring Based on a Convolutional Neural Network.* Sensors. 23. 8525. 10.3390/s23208525.

[6] Eduardo M. Coraça, Janito V. Ferreira, Eurípedes G.O. Nóbrega, *An unsupervised structural health monitoring framework based on Variational Autoencoders and Hidden Markov Models*, Reliability Engineering and System Safety, https://doi.org/10.1016/j.ress.2022.109025.

[7] Niklas Römgens, Abderrahim Abbassi, Clemens Jonscher, Tanja Grießmann, Raimund Rolfes, *On using autoencoders with non-standardized time series data for damage localization* Engineering Structures, https://doi.org/10.1016/j.engstruct.2024.117570.

[8] Luca Benfenati and Daniele Jahier Pagliari and Luca Zanatta and Yhorman Alexander Bedoya Velez and Andrea Acquaviva and Massimo Poncino and Enrico Macii and Luca Benini and Alessio Burrello *Foundation Models for Structural Health Monitoring*, 2024, https://arxiv.org/abs/2404.02944

[9] Po-Yao Huang and Hu Xu and Juncheng Li and Alexei Baevski and Michael Auli and Wojciech Galuba and Florian Metze and Christoph Feichtenhofer *Masked Autoencoders that Listen*, https://arxiv.org/abs/2207.06405

[10] Najafi, Amirali Braley, John Gucunski, Nenad Maher, Ali. (2023). *Generative adversarial network for predicting visible deterioration and NDE condition maps in highway bridge decks.* Journal of Infrastructure Intelligence and Resilience. 2. 10.1016/j.iintel.2023.100042.

[11] Ballester C, Bertalmio M, Caselles V, Sapiro G, Verdera J. *Filling-in by joint interpolation of vector fields and gray levels.* IEEE Trans Image Process. 2001;10(8):1200-1211. doi:10.1109/83.935036

[12] Bradski, G. (2000). *The OpenCV Library.* Dr. Dobb's Journal of Software Tools.

[13] Telea, Alexandru. (2004). *An Image Inpainting Technique Based on the Fast Marching Method.* Journal of Graphics Tools. 9. 10.1080/10867651.2004.10487596.

[14] Diederik P Kingma and Max Welling *Auto-Encoding Variational Bayes*, https://arxiv.org/abs/1312.6114

[15] Jim Nilsson and Tomas Akenine-Möller *Understanding SSIM*, https://arxiv.org/abs/2006.13846

[16] Ilya Loshchilov and Frank Hutter *Decoupled Weight Decay Regularization*, https://arxiv.org/abs/1711.05101

[17] Ilya Loshchilov and Frank Hutter *SGDR: Stochastic Gradient Descent with Warm Restarts*, https://arxiv.org/abs/1608.03983

[18] Ian J. Goodfellow and Jean Pouget-Abadie and Mehdi Mirza and Bing Xu and David Warde-Farley and Sherjil Ozair and Aaron Courville and Yoshua Bengio *Generative Adversarial Networks*, https://arxiv.org/abs/1406.2661