Corrosion-Induced Cracking Indication in Reinforced Concrete

James Wong
jmswong@stanford.edu

Melissa Zirps
mzirps@stanford.edu

I. Introduction

We train models to predict corrosion-induced cracking, which can then be used in digital twinning [1]. The models take in corrosion patterns and concrete material properties and outputs whether there is a crack on the concrete's surface. We use a combination of convolutional neural networks and fully-connected neural networks to handle our multi-modal inputs.

There is a need for deep learning here because although existing modelling techniques are accurate, they are too computationally expensive, taking 12hrs to run on a 5cm diameter sample. Concrete is usually repaired when cracking (usually driven by corrosion) reaches the surface; by being able to predict this quickly at a large scale, the need for tedious visual inspection of large bridges/buildings is negated.

II. Related Work

To our knowledge, there has been no prior work using corrosion patterns as inputs for any task, as well as no prior work predicting surface cracking from any inputs [1]. The closest similar work was done by Boukhatem et al., which uses only 90 experimental data points to predict time-to-cracking based on concrete material properties [2]. Several other projects use concrete material properties to predict other outputs, including compressive strength [3], carbonation depth [4], and rebar corrosion level [5].

Parts of our work take inspiration from projects outside of civil engineering:

We use 1D convolution to learn from corrosion depth inputs. In the literature, 1D convolution is commonly applied to time-series [6], signal processing [7], and hyper-spectral imaging [8]–[10] applications.

Our model combines two different types of input features (corrosion patterns and concrete properties). One method to handle multimodel input data is concatenating features or feature representations, then feeding the result into deeper layers. Miller et al. combined images and text for image classification [11]. Venugopalan et al. combined MRI images, genetic sequences, and clinical test data for detection of Alzheimer's disease stage [12].

III. Dataset

A. Dataset Overview

The dataset consists of two types of input features:

[1]A digital twin is a virtual model of a real-world physical object

1) A vector of corrosion depths in $\mathbb{R}^{337x1}$, representing the amount of corrosion along the z-axis of a vertical cylindrical rebar. We use a 1D representation since corrosion at two points at the same height (z) but different angle (θ) are usually very similar, relative to corrosion between two points at different heights.

2) Four additional continuous features representing properties of the concrete itself:

- Rebar Radius: The radius of the single reinforcing bar through the concrete cylinder, in meters.
- Concrete Cover: Thickness of the concrete covering the reinforcement, in meters.
- Tensile Strength: The strength of the concrete in tension, in mega-pascals.
- Water-to-cement Ratio: The proportion of water to cement in the concrete mix.

Figure 8 (appendix) shows the signal distributions for these signals, and provides some intuition on how they relate to concrete cracking.

B. Data Generation

To generate the data, first we generate corrosion patterns using COMSOL multiphysics, a finite element analysis, solver, and simulation software package. To allow for faster data generation, it was assumed that corrosion is constant radially and only varies along the length of the reinforcement.

Once the corrosion patterns have been determined, they are inputted into the matlab coupled finite element lattice model (FEM). The coupled model first assigns random parameters to the four additional continuous features (rebar radius, concrete cover, tensile strength, and water-to-cement ratio). Then it meshes the concrete cylinder and applies the corrosion load to determine the cracking pattern. From the crack pattern, we compute the binary label for whether any crack has reached the surface. Figure 3 shows a visualization of this pipeline.

This process is computationally expensive, so data generation has been a slow process, taking around 12 hours to generate a single training example. To generate our dataset, we had to run this pipeline continuously across multiple machines. At the time of writing, our dataset contained 1,982 samples.

C. Data Preprocessing

After running COMSOL and FEM to generate the data, we extract the outputs from the simulations, process/transform

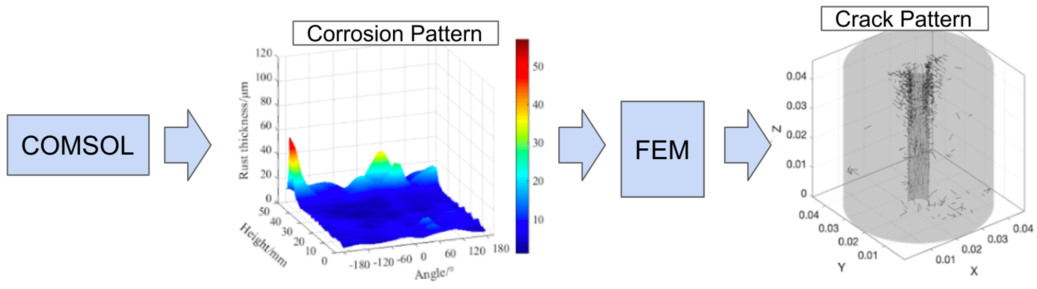

Fig. 1. Input (corrosion pattern) and Output (crack pattern) of FEM.

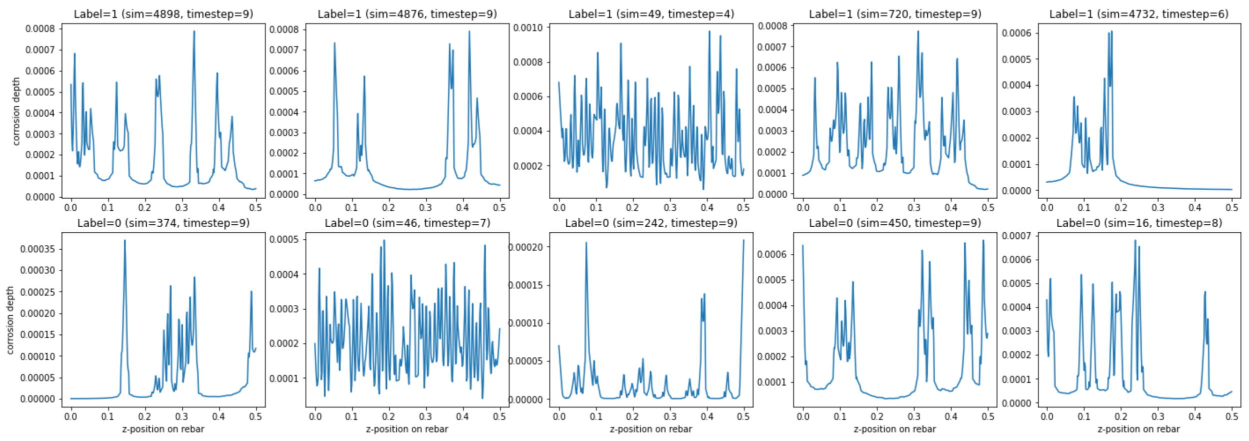

Fig. 2. Example positive (top) and negative (bottom) corrosion patterns, represented as 337x1 dimensional vectors.

corrosion patterns, join the corrosion patterns with the corresponding concrete features, and extract/append the target labels.

D. Data Analysis

The dataset consists of 37.35% positive (surface cracking) and 62.65% negative samples. In the real world, positive samples would be much more rare. However, for our dataset we chose the data generation parameters to dramatically upsample positive cases.

A typical corrosion pattern has a low baseline corrosion level, with concentrated regions of high corrosion. Figure 2 shows some example corrosion patterns in 1D representations.

Although individual samples are quite erratic, in general rebars with more corrosion are more likely to see concrete cracking. This is observed in our training data and shown in Figure 3.

IV. Learning Methods

A. Data Splitting

We split the data into 60% train / 20% validation / 20% test sets. Due to the scarcity of data, after selecting hyperparameters with the validation set, we train one additional model on the combined train+validation data (80%) before reporting results on the test set. This yielded a moderate improvement on test set performance compared to just training on the 60% train set.

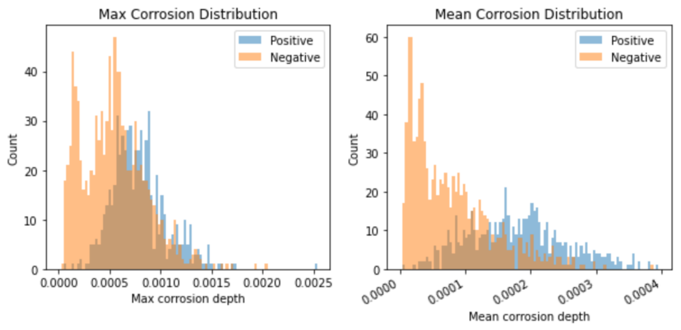

Fig. 3. Maximum and mean corrosion depth distributions.

B. Training Objective

Early on, our training set was heavily biased towards negative (no surface crack) examples, so we trained with a weighted binary cross-entropy (BCE) loss with higher weights on positive samples:

$$L(y,\hat{y}) = -\frac{1}{n_x}\sum_{i=1}^{n_x}(\alpha \cdot y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})) \quad (1)$$

We set $\alpha = 1$ in the baseline model. Later on, we addressed the data imbalance by modifying the data generation pipeline to upsample concrete properties that were more likely to result in cracking, and switched to an unweighted BCE loss ($\alpha = 1$).

C. Baseline Model Architecture

For our baseline model, which we denote Conv1-FC1, we first pass the corrosion depth input through a 1D convolution with a single kernel of size 20x1, followed by ReLU and max pooling with a kernel and stride of 16. This reduces the (337 x 1) corrosion depth input to a (19 x 1) vector. This is then concatenated with the (4 x 1) vector of concrete properties, and then passed through a fully-connected layer with one output node. Finally, a sigmoid activation is applied to get the final model prediction.

Figure 4 is a graphical representation of this model:

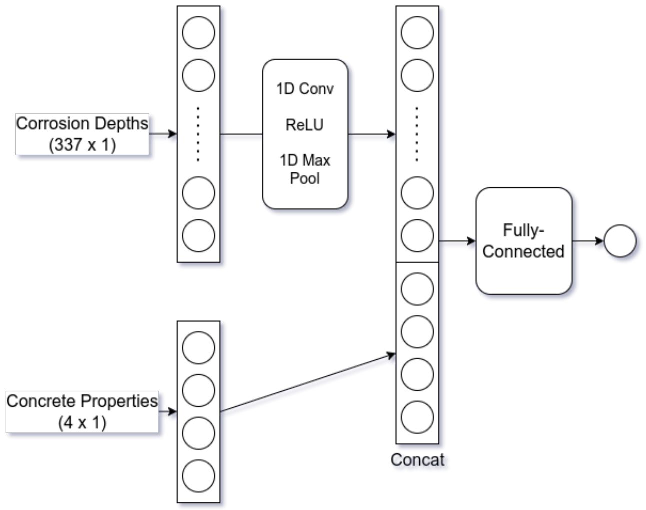

Fig. 4. Conv1-FC1 Network Architecture

Table I shows the performance of the baseline. When we first trained the baseline model, we had only generated 220 training samples, so we trained the baseline model on the entire dataset at the time. Test set metrics were added retroactively. We observe the small dataset resulted in overfitting.

TABLE I
BASELINE MODEL PERFORMANCE

	Train Loss	Train F1 Score	Test Loss	Test F1 Score
Conv1-FC1	0.220	0.935	0.777	0.543

D. Increasing Dataset Size

The largest gains in model performance came from adding more training data. Table II shows the effect of dataset size on model performance for the baseline model. A similar behavior was observed for all other models we tried. The last row in Table II was trained on the train+validation combined set.

TABLE II
EFFECT OF DATASET SIZE ON BASELINE MODEL

Dataset Size	Train Loss	Train F1 Score	Test Loss	Test F1 Score
220	0.220	0.935	0.777	0.543
660	0.301	0.850	0.355	0.819
1268	0.280	0.865	0.316	0.851
1585	0.306	0.885	0.314	0.865

E. Data Normalization

In our baseline model, we used the standard feature normalization technique, where we compute the mean and standard deviation for each input feature independently, and then scale each feature to have 0 mean and 1 variance.

$$\tilde{x_i} = \frac{x_i - \mu_i}{\sigma_i} \tag{2}$$

Although this works reasonably well for concrete property inputs, this performs poorly on corrosion depth inputs. Instead we implemented custom normalization, where the mean and standard deviation was computed across the entire corrosion depth vector across *all* samples. Then all points were scaled by these values. This type of normalization resulted in significantly improved model performance. Table III shows the effect of this normalization on the Conv1-FC1 (baseline) model architecture, trained on 1,268 training samples.

TABLE III
EFFECT OF DATA NORMALIZATION

	Train Loss	Train F1 Score	Test Loss	Test F1 Score
Standard Normalization	0.578	0.556	0.540	0.615
New Normalization	0.280	0.865	0.316	0.851

F. Deeper Models

With the larger datasets, the baseline model showed low variance. We attempted to decrease bias by adding more layers to the model. We tried several model architectures following a similar template: Corrosion patterns are passed through one or more layers of convolution blocks (1D Convolution → ReLU → 1D MaxPool). Concrete properties are passed through one or more fully-connected layers. The two outputs are concatenated and then passed through one or more fully-connected layers. Figure 5 shows this general model architecture.

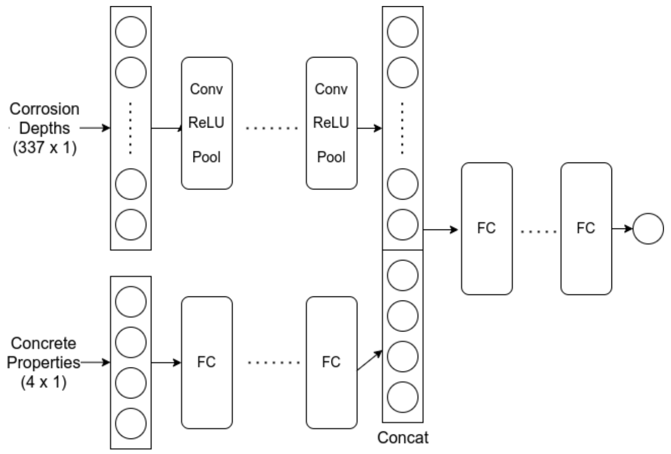

Fig. 5. General Network Architecture

We use the following naming convention: If $\mathbf{X}$ is the number of convolution blocks, $\mathbf{Y}$ is the number of fully connected layers preceding concatenation, and $\mathbf{Z}$ is the number of fully connected layers after concatenation, we denote this model as Conv$\mathbf{X}$−H$\mathbf{Y}$−FC$\mathbf{Z}$ ("H" for "hidden fully connected").

We experimented with multiple model architectures, and for each architecture, we used random search to tune hyperparameters such as convolution/pooling kernel sizes, convolution/pooling strides, and fully-connected layer sizes. Interestingly, most of these deeper models did not decrease training loss, and thus yielded no improvements on test set performance. One possible explanation is that we are close to the optimal model given the current dataset (i.e. most of the remaining losses are mistakes in simulation resulting in mislabled samples). If so, the only way to improve performance is to collect more and/or higher-quality training data.

TABLE IV
CONV1 - H1 - FC1 MODEL PERFORMANCE

Conv Kernel Size	Pool Stride	Hidden Layer Size	Train Loss	Train F1 Score	Test Loss	Test F1 Score
8	8	4	0.243	0.881	0.297	0.853
8	8	8	0.237	0.890	0.246	0.878
8	8	16	0.240	0.891	0.304	0.848

TABLE V
CONV1 - H1 - FC2 MODEL PERFORMANCE

Output FC Layer Sizes	Train Loss	Train F1 Score	Test Loss	Test F1 Score
8, 1	0.234	0.890	0.294	0.857
16, 1	0.230	0.881	0.255	0.884
32, 1	0.238	0.880	0.301	0.853
64, 1	0.237	0.879	0.296	0.854
128, 1	0.233	0.889	0.261	0.850
256, 1	0.234	0.886	0.297	0.854

TABLE VI
CONV2 - H1 - FC1 MODEL PERFORMANCE

Conv Kernel Sizes	Output FC Layer Sizes	Train Loss	Train F1 Score	Test Loss	Test F1 Score
8, 8	1, 8	0.257	0.871	0.281	0.836
8, 8	1, 4	0.241	0.881	0.331	0.838
16, 8	1, 8	0.245	0.884	0.293	0.857
8, 8	4, 2	0.268	0.835	0.269	0.853
8, 8	4, 2	0.251	0.878	0.254	0.873

G. Data Augmentation

Since model performance seemed strongly correlated with dataset size, we tried to artificially increase the dataset size via data augmentation. We implemented 3 types of data augmentation:

- Flipping: Since corrosion patterns are symmetric and none of the concrete property features depend on the corrosion pattern, we can flip the 1D corrosion vector.
- Adding noise: We can add a small random Gaussian noise to all input features independently.
- Monotonic scaling: This utilizes the fact that increasing corrosion (multiplying the vector by some constant > 1) can only increase the likelihood of cracking, and vice versa. We use the existing label to either scale the corrosion depth inputs up or down, while maintaining correctness of the label.

We generated 10,000 additional samples by randomly sampling from the above 3 categories. Experimental results (Table VII) did not show an improvement in performance from data augmentation. This could have been due to poor tuning of data augmentation parameters, such as adding too much noise or scaling too much.

TABLE VII
DATA AUGMENTATION ON CONV1-H1-FC1 MODEL

	Train Loss	Train F1 Score	Test Loss	Test F1 Score
Normal Data	0.265	0.859	0.259	0.868
Augmented Data	0.239	0.870	0.260	0.851

H. Hyperparamter Tuning

We use RayTune [13], [14] for parallelized asynchronous hyperparameter search. For each architecture, we train 100 models with random hyperparameters, trained on the 60% training set, and select the best hyperparameters based on the 20% validation set. Using a classification threshold of 0.5, we measure validation loss, precision, recall, f1 score, and AUC for each model. Experimental results showed that, with the exception of loss, the other metrics mostly are mostly aligned on which model is better. We chose to maximize f1 score as the hyperparameter tuning objective.

We tried two search schedulers.

- Asynchronous Successive Halving Algorithm (ASHA) [15] speeds up the search by aggressive early stopping of low-performing runs.
- Population Based Training (PBT) [16] chooses the next hyperparameters to try based on results from previous runs, as opposed to randomly.

Since this pipeline is computationally expensive, we only ran full searches for a handful of the model architectures we tried. For the remaining models, we set the hyperparameters based on smaller searches and results from earlier experiments. Table VIII shows the hyperparameter ranges we searched over for one model architecture, and the best hyperparameters.

TABLE VIII
BEST HYPERPARAMETERS FOR CONV1-H1-FC1 MODEL

Hyperparameter	Search Range	Selected Hyperparameters
Learning Rate	Log Uniform $(10^{-4} : 10.0)$	0.0035
L2 Regularization Amount	Log Uniform $(10^{-5} : 10^{-2})$	0.0134
Optimization Algorithm	Adam, RMSprop, SGD	RMSprop
Batch Size	64, 128, 256, 512	128
Convolution LayerKernel Size	8, 16, 32	32
Pooling Stride	4, 8, 16	4
Hidden Layer Size	8, 16, 32	8

I. Transfer Learning

Since we had access to a dataset of 84,834 corrosion pattterns (no concrete property features, and no labels), we had initially planned to use transfer learning, to learn a dense representation of corrosion patterns and apply it to our supervised learning application. However, after processing the unlabeled dataset, we noticed it had a significantly different distribution from our labeled data, as shown in Figure 9 (appendix), so we did not pursue this further.

V. Results and Discussion

All of the model architectures we tried, including the baseline architecture, had similar results when trained on our largest training dataset with custom feature normalization. Our best performing model, by a small margin, was Conv1-H1-FC1 Model with the parameters specified above, with a test F1 score of 0.891. Table IX contains detailed metrics for this model.

TABLE IX
Conv1-H1-FC1 test set metrics

Test Accuracy	0.922
Test Precision	0.927
Test Recall	0.858
Test F1 Score	0.891
Test AUC	0.955

Confusion Matrix		
	Prediction= Positive	Prediction= Negative
Label= Positive	127 (31.99%)	21 (5.29%)
Label= Negative	10 (2.52%)	239 (60.20%)

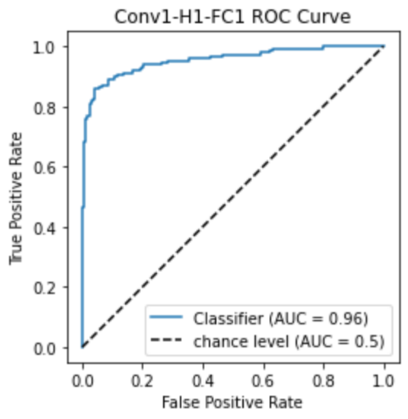

Fig. 6. ROC Curve

For most of our project, we used a fixed classification threshold of 0.5 to make a binary classification decision from the model output. For this application, we prefer a high-recall model, since false negatives (failing to identify a crack) pose a greater risk than false positives (requiring visual inspection). We can increase recall at the cost of precision through tuning the classification threshold. Figure 6 shows the ROC curve for this model. Table X shows test set metrics with an aggressively lower threshold. The results are quite promising; we can achieve near-perfect recall and still maintain 44.5% precision. If the test set metrics generalizes to real-world data, we would be able to identify almost all surface cracks, and only require a bit more than twice as many inspections compared to a perfect (100% accuracy) model.

TABLE X
Conv1-H1-FC1 Model with Classification Threshold=0.01

Test Accuracy	0.537
Test Precision	0.445
Test Recall	0.993
Test F1 Score	0.651

Confusion Matrix		
	Prediction= Positive	Prediction= Negative
Label= Positive	147 (37.03%)	1 (0.25%)
Label= Negative	183 (46.10%)	66 (16.62%)

A. Qualitative Results

To qualitatively evaluate the results, visualizations of the cracking patterns were created for some false negative (Figure 10, appendix) and false positive (Figure 11, appendix) examples. It was observed that in almost all cases where a false negative occurred, the crack barely made it to the surface. On the other hand, false positives contain multiple cracks very close to the surface, without actually reaching the surface.

The model was also tested on real reinforced concrete samples from experimental data, as shown in Figure 7. We had 3 such experimental samples- two positive and one negative. Inputting this data into our best model resulted in all examples being classified as having no surface cracking (33% accuracy). It was determined this was because the data from these experiments was outside of the distribution that the model was trained on.

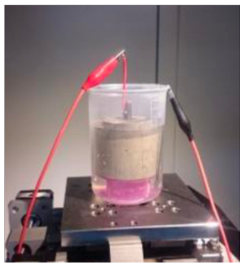

Fig. 7. Experimental Sample

VI. Conclusion / Future Work

In this project, we trained a multi-model model which predicts surface cracking based on corrosion patterns on a rebar, and properties of the surrounding concrete. Since large datasets for concrete cracking do not exist, we collected training data through simulations. We used custom data normalization, tried several model architectures, and used random search to tune hyperparameters. The largest model improvements came from data normalization and increasing training dataset size.

We observed that the amount of data played a big role in model performance, so as future work, we will continue to generate more data from simulations, and expand the range of inputs so that the model can be applied to less common inputs, such as the experimental sample we tested on.

Contributions

James: Implemented data preprocessing/normalization and model training pipelines. Implemented random search and tuned model hyperparameters. Contributed to written report.

Melissa: Wrote the data generation code and tweaked it to try and produce balanced data. Contributed to written report.

Implementation

https://github.com/jmswong/concrete_corrosion

References

[1] Woubishet Zewdu Taffese and Esko Sistonen. Machine learning for durability and service-life assessment of reinforced concrete structures: Recent advances and future directions, 5 2017.

[2] Ablam Zidol and Arezki Tagnit-Hamou Bakhta Boukhatem. Prediction of time-to-corrosion cracking of reinforced concrete using deep learning approach. *ACI Symposium Publication*, 349, 2022.

[3] Ahmed M. Diab, Hafez E. Elyamany, Abd Elmoaty M. Abd Elmoaty, and Ali H. Shalan. Prediction of concrete compressive strength due to long term sulfate attack using neural network. *Alexandria Engineering Journal*, 53:627–642, 9 2014.

[4] Hae-Chang Cho, Hyunjin Ju, Jae-Yuel Oh, Kyung Jin Lee, Kyung Won Hahm, and Kang Su Kim. Estimation of concrete carbonation depth considering multiple influencing factors on the deterioration of durability for reinforced concrete structures. *Advances in Materials Science and Engineering*, 2016:1–18, 2016.

[5] Yidong Xu and Ruoyu Jin. Measurement of reinforcement corrosion in concrete adopting ultrasonic tests and artificial neural network. *Construction and Building Materials*, 177:125–133, 7 2018.

[6] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151, 2021.

[7] Serkan Kiranyaz, Turker Ince, Osama Abdeljaber, Onur Avci, and Moncef Gabbouj. 1-d convolutional neural networks for signal processing applications. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8360–8364, 2019.

[8] Haitao Zhang, Lingguo Meng, Xian Wei, Xiaoliang Tang, Xuan Tang, Xingping Wang, Bo Jin, and Wei Yao. 1d-convolutional capsule network for hyperspectral image classification. *CoRR*, abs/1903.09834, 2019.

[9] Xiangpo Wei, Xuchu Yu, Bing Liu, and Lu Zhi. Convolutional neural networks and local binary patterns for hyperspectral image classification. *European Journal of Remote Sensing*, 52(1):448–462, 2019.

[10] Aswathi Soni, Mahmoud Al-Sarayreh, Marlon M. Reis, and Gale Brightwell. Hyperspectral imaging and deep learning for quantification of clostridium sporogenes spores in food products using 1d- convolutional neural networks and random forest model. *Food Research International*, 147:110577, 2021.

[11] Stuart J Miller, Justin Howard, Paul Adams, Mel Schwan, Robert Slater, Stuart J ; Miller, Justin ; Howard, Paul ; Adams, Mel ; Schwan, and Stuart Miller. Multi-modal classification using images and text, 2020.

[12] Janani Venugopalan, Li Tong, Hamid Reza Hassanzadeh, and May D. Wang. Multimodal deep learning models for early detection of alzheimer's disease stage. *Scientific Reports*, 11, 2021.

[13] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[14] Sovit Ranjan Rath. Hyperparameter tuning with pytorch and ray tune, Dec 2021. https://debuggercafe.com/hyperparameter-tuning-with-pytorch-and-ray-tune/.

[15] Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. *ICLR*, 2018.

[16] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

Appendix

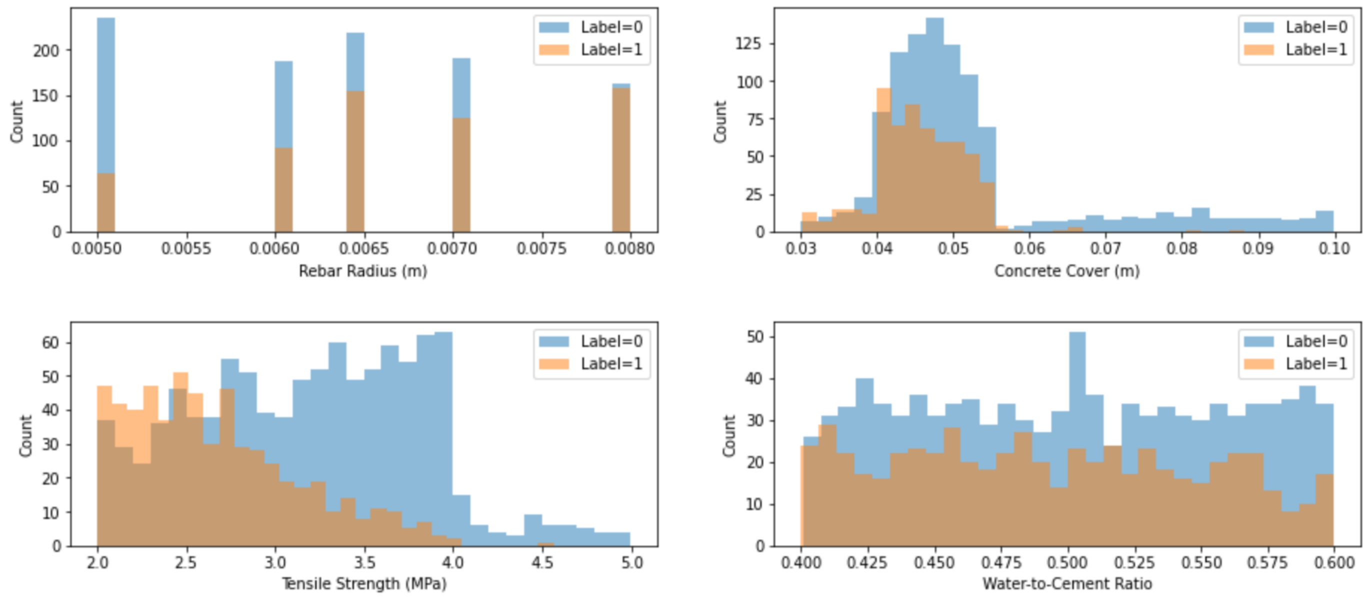

Fig. 8. Concrete property signal distributions.

Rebar Radius: Larger rebar radius can result in cracking initiating sooner.
Concrete Cover: Generally, thicker concrete cover is less likely to result in surface cracking.
Tensile Strength: Cracking occurs when the internal pressure of the corrosion product exceeds the tensile capacity of the concrete, so a lower tensile strength results in more extensive cracking.
Water-to-Cement Ratio: A higher water to cement ratio results in a more porous cement matrix, which can absorb some of the corrosion product, lessening the internal pressure and subsequently reducing cracking.

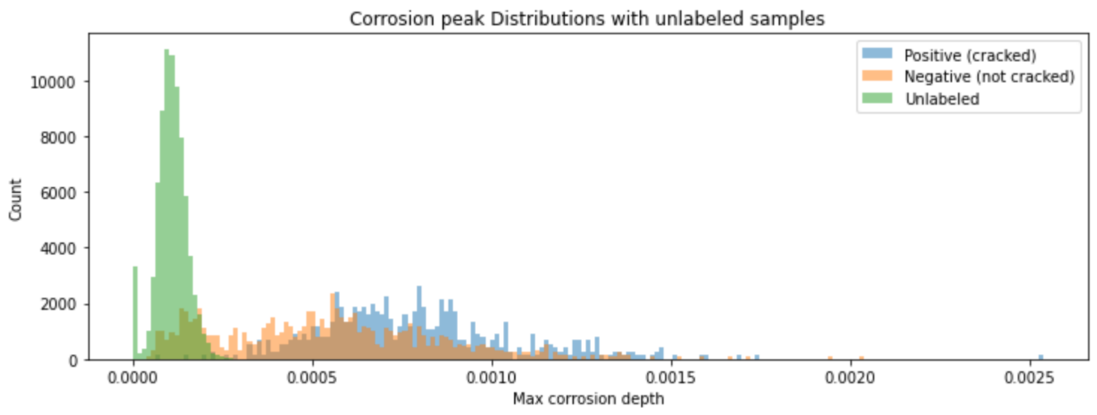

Fig. 9. Max corrosion depths for unlabeled, positive, and negative samples

The unlabeled dataset (green) had much lower corrosion depth values compared to both the positive and negative labeled examples. We also observed a significant number unlabled examples which had almost 0 corrosion at every point.

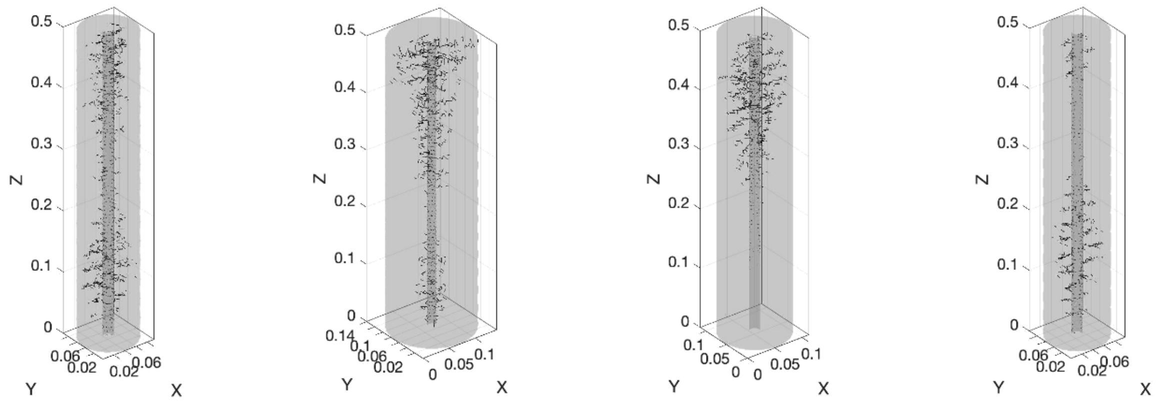

Fig. 10. False Negative Examples
False negative examples usually show moderate cracking at a single small area of the concrete. For many examples, internal cracking is minimal on most other regions of the concrete.

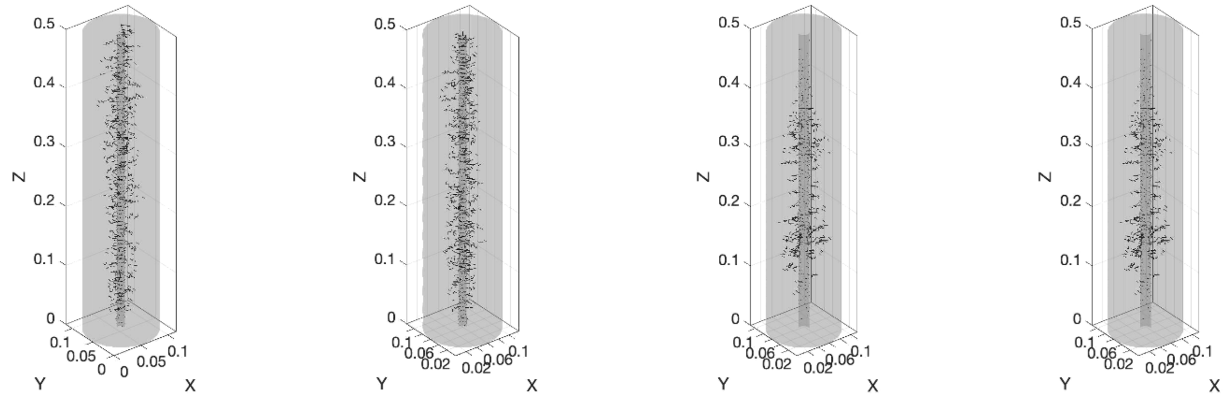

Fig. 11. False Positive Examples
False positive examples show significant internal cracking, even though no one crack made it to the surface.