Automatic Crack Detection Using Encoder-Decoder Architecture

Nathan Loh, Nikolaus Schultze
Department of Computer Science
Illinois Institute of Technology

April 17, 2023

Abstract

This is a report for the project in cs512. In this project, we will create an alternative method for properly monitoring concrete maintenance to detect and prevent cracks. Our attempt will include creating a U-Hierarchical Dilated Network (U-HDN) from scratch. After creating and training the model, we will test its performance using Recall, Precision, F1-score, ODS, and OIS. In addition to the U-Hierarchical Dilated Network, we will use alternative models such as a simple Convolutional Neural Network and an encoder-decoder network. We will test these models' performances and compare their results to our U-Hierarchical Dilated Network. The link for the Google Drive containing the data and trained models:
https://drive.google.com/drive/u/2/folders/1NaMlcS0X7TWlTLj0BzAtMUpojXAZ_ssy.

1 Problem Statement

Concrete cracking can pose several risks to humans, their property, and other animals. Cracks resulting in uneven surfaces can be a tripping hazard causing harm to the individual. They can also cause damage to vehicles, such as scraping the tires or underbelly of the vehicle, which can severely damage the car and cause a further collision that can potentially cause more harm to humans. Cracks can also allow mold to grow if water can settle in the cracks. This mold could harm humans, animals, and the surrounding environment. Concrete cracking can also result in falling debris. Falling debris can have profound effects depending on its location: the potential to fall on someone in a building and the potential for vehicles to be struck by debris falling from a bridge onto a road below, which can result in severe crashes. These cases can lead to severe injuries and even death. Several factors, including stress from heavy loads, water intrusion, wear and tear, and shrinkage over time, can cause concrete cracking. Concrete cracking can be prevented with regular maintenance and proper monitoring; however, many areas cannot afford the cost related to this maintenance and monitoring. As a result, their concrete will be at a higher risk of cracks, which can harm the people and animals in that area.

2 Proposed Solution

Our solution to this problem is to implement an object detection algorithm. The object detection algorithm would serve as the proper maintenance required to combat concrete cracking and requires images of concrete. The algorithm would then detect whether or not the concrete in the image contains a crack. If the algorithm determines that the concrete contains a crack, it will allow the users to know and result in the concrete in the image receiving the maintenance it requires. One of the algorithms we would utilize is the algorithm proposed in the paper by Fan, Li, Chen, and Whei. The proposed algorithm is a U-Hierarchical Dilated Network (U-HDN), which utilizes an encoder-decoder algorithm to perform crack detection. The algorithm consists of three components: a U-net architecture to extract the features and restore the image, a multi-dilation module to get more information about the features, and a hierarchical feature learning module to obtain multi-scale features. Our dataset contains two folders, Positive, which contains 20,000 images of concrete containing cracks, and Negative, which contains 20,000 images of concrete that do not contain cracks. The data will be split into training, validation, and testing groups before feeding into the network. The model will be trained on parts of the dataset, and when tested by inputting images, it will state whether there are cracks or not. To evaluate the model's performance, we will use the validation data to see how accurate the model is at determining whether or not the concrete surface has cracks. We will evaluate the

model's Recall, Precision, F1-score, ODS, and OIS. We will also implement different algorithms, such as a simple Convolutional Neural Network, a complex Convolutional Neural Network, and an encoder-decoder network, to detect cracks in the dataset. We will use the results of these alternative models and compare their performances with the performance we got from training and testing the U-HDN algorithm.

The components of the U-HDN can be represented in Figures 1 and 2. The U-net architecture is the main structure of the U-HDN algorithm. It is composed of two parts: the encoder and the decoder. The purpose of these parts is to first extract features at different spatial resolutions before reconstructing the original image through upsampling. The encoder consists of blocks made of two 3 by 3 convolution layers with a batch normalization and a dropout layer in between, and then a max pooling layer of 2 by 2 at the end to reduce the image size. Right before the max pooling layer, there is another batch normalization layer. The decoder consists of blocks of a 2 by 2 up-convolution that acts as upsampling layers, two 3 by 3 convolution layers that also have a batch normalization and dropout layer in between the layers and a batch normalization layer after the second convolutional layer. It is important to note that the fourth pooling layer, ninth and tenth convolution layers, and the first up-convolution layer were removed. For all convolutional layers, it uses the ReLU activation function and zero-padding. Skip connection is also implemented in the decoder section, where the upsampled image is concatenated with its corresponding layer in the encoder.

The multi-dilation module is composed of four separate double 3 by 3 convolution layers with different dilation rates of 2, 4, 8, and 16. Convolution layers that are connected will have the same dilation rate but are different from the layers of separate paths. Once all four outputs are made, it is then concatenated together before being passed into a 1 by 1 convolution layer to resize it to 1024.

The last section of the U-HDN is the hierarchical feature learning module. In this section, we need to extract the high-level feature maps at the high and low convolutional levels. We do this by creating multiple convolutional layers with different kernel sizes of 1, 3, and 5, as well as different dilation rates of 2, 4, and 8. We then concatenated these while having them have an activation function of ReLU (different from the paper) to be used to detect if there are cracks. For the final output, we used a loss function of sparse categorical cross-entropy and the optimizer of Adam.

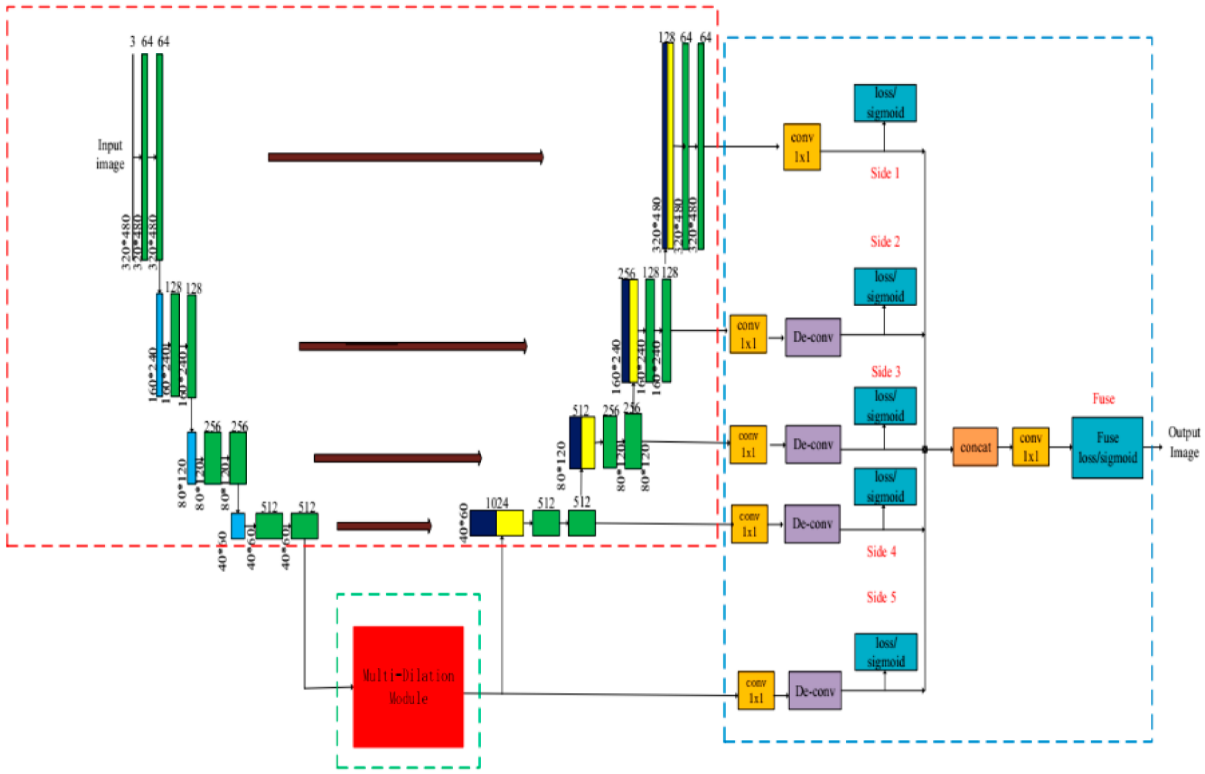


Figure 1. The U-HDN architecture separated into three sections: U-net architecture, multi-dilation module, and hierarchical feature learning module. The red dotted box represents the U-net architecture, the green dotted box indicates the multi-dilation module, and the blue dotted box stands for the hierarchical feature learning module.

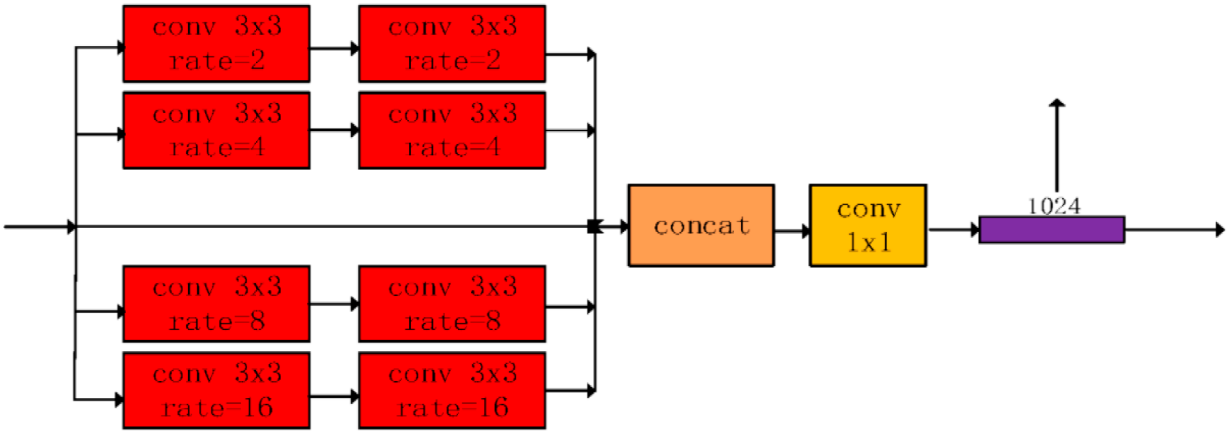


Figure 2. The multi-dilation module overview composed of convolutional layers with different dilation rates before being concatenated and passed into a 1 by 1 convolution filter.

3 Implementation Details

Program Design Issues:

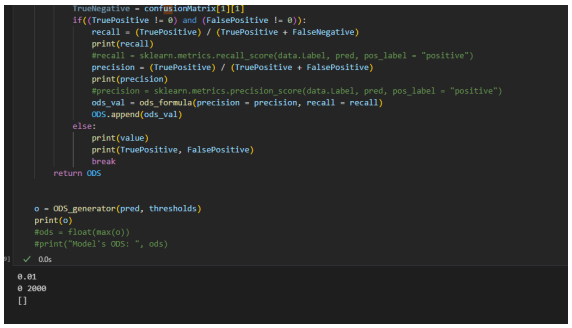
When creating the U-HDN algorithm, we faced a design issue: building the Multi-Dilation Module and Hierarchical Feature learning module. When building these parts, we ran into problems with how to build them exactly as indicated by the paper. Specifically, some details were not provided, such as how many filters would be in the convolution layers. Another problem while designing the network was the logits and label shape error. We used sparse categorical cross-entropy, which required the logits and label shapes to be based on [batch_size, output_shape, num_classes] and [batch_size, num_classes]. However, we constantly had an error where the logits shape was [batch_size*outputshape, num_classes] while the label shape was only [batch_size]. We checked the designs of the encoder, decoder, multi-dilation module, and hierarchical feature learning module and saw it was an output layer error. Due to time constraints, we used a less efficient method of flattening the output layer and applying a dense layer with two neurons.

Problems Faced and Their Solutions:

One problem we faced was when calculating the OIS of the models. The equation to calculate OIS can be found in the paper and can be shown below:

$$\mathrm{OIS} = \frac{1}{N_{img}} \sum_{i}^{N_{img}} max \frac{2 \times Pr_t^i \times Re_t^i}{Pr_t^i + Re_t^i} : t = 0.001, 0.002, \ldots, 0.999$$

We implemented this equation into the code to calculate each model's OIS. However, we were given several scalar errors after running the code for a while. When the code finally finished calculating, we were given low values for all the models. In the paper, the tested models received an OIS ranging from 0.6 to 0.9, so we believe our value was invalid. We believe these scalar errors and the long run time occurred because the function was too complex and the computer did not have the capacity of being able to run it. We attempted to find solutions to this problem online but were unable to find a solution. We contacted the TA, informed him of this problem, and asked for possible alternatives to run the code.



Another issue we faced was when this if statement failed. The if statement's purpose was to efficiently determine the F1-Score for each threshold. The if statement should fail when both TruePositive and Negative equal 0, without this if statement, the function would have calculated 0 for potentially dozens of thresholds which would have been inefficient. However, it failed when TruePositive equals 0, and FalseNegative equals 2000. We removed this if statement for the model that it failed on, and the function ran correctly, and we received a value for ODS. However, we received scalar errors due to the formula trying to divide 0 by a number.

Another problem faced was time. Because we had multiple large networks, some took less than a minute per epoch, and others took almost twenty minutes per epoch. As a result, we had to use fewer epochs for the networks with epochs that took longer to train, resulting in the model not accurately representing the network. We had tried to run the networks on other GPU services such as Google Colab but failed to successfully train due to the site constantly crashing due to memory problems and limited GPU provided.

Instructions:

All of the models are already trained and saved. Download the trained models and the training, validation, and testing datasets and move them to the same folder as the program files. Make sure to unzip the training dataset before running the program. Before running the program, change the directory of the datasets to the respective location in each program file. To run each model, run the Import Libraries, Getting Image Paths, Labeling Data, and Generating Data cells. You will now be able to load the already trained models and their history by running the Loading Trained Model cell. After loading the model and its history, you will be able to run the cells to evaluate the performance of the model and make a prediction on an inputted image.

4 Results and Discussions

Performance Evaluation Metrics Used:

- Recall: measures the model's ability to detect Positive samples (completeness of positive predictions). The ratio of the number of Positive samples correctly predicted as Positive and the total number of Positive samples.
- Precision: measures the model's accuracy in classifying a sample as positive (accuracy of positive predictions). The ratio of the number of Positive samples correctly predicted and the total number of samples classified as Positive.
- F1-Score: measure's the model's accuracy through class-wise performance using Precision and Recall.
- ODS: the best F1 on the public database for a fixed threshold.
- OIS: the aggregate F1 on the public database for the best threshold in each image.
- Accuracy: ratio of the number of correct predictions to the total number of predictions.

U-Hierarchical Dilated Network

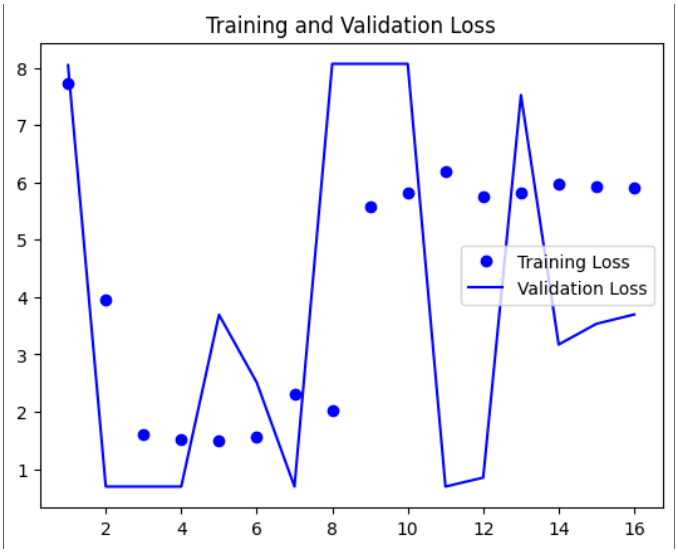
Recall	0.396
Precision	0.496

F1-Score	0.440
ODS	0.597
OIS	0.025 (Invalid)
Accuracy	0.497

The model's Recall score was 0.396, which means the model predicts under half of the total positive samples as positive, meaning that the model has a smaller number of True Positive samples as False Negative Samples. The model's Precision score was 0.496, which means about half of the samples the model predicts to be positive are positive, meaning the model has a similar number of True Positive samples as False Positive samples. The model's F1 Score was 0.440, and the model's Accuracy was 0.497. Both of these mean the model was less than 50% accurate at predicting the correct result. The model's ODS was 0.597, meaning there is a better threshold for this model than 0.5. We could increase the ODS by changing the threshold value from 0.01 to 0.001 to find the best threshold and get the highest ODS possible, which would also get us the highest F1-Score. Changing the threshold to this new threshold would increase the model's Accuracy. The model's OIS was 0.025; however, we believe this is invalid as it is a minimal value, and the OIS values they got in the paper ranged from 0.6 to 0.9. Overall, this model was not very good at predicting positive and negative samples, as shown by the low Recall, Precision, F1-Score, and Accuracy, and is only right about half of the time. However, the ODS is relatively high, meaning that this model can be almost 60% accurate with the correct threshold. It is important to note that we ran the U-HDN model with 16 epochs, so increasing the number of iterations may drastically improve the accuracy.

Positive Recall	0.50
Negative Recall	0.50
Positive Precision	0.40
Negative Precision	0.60
Positive F1-Score	0.44
Negative F1-Score	0.54

As we can see, the model had similar results for predicting Positive and Negative. The Positive and Negative Recall of the model is 0.50, meaning that the model managed to predict about 50% of the Positive and Negative samples. However, the model's Negative Precision is 0.60, whereas its Positive Precision is only 0.40. This means the model was more accurate when predicting Negative than Positive. This would result in the Negative F1-Score being higher than the Positive F1-Score, and we see the Negative F1-Score is 0.54 and the Positive F1-Score is 0.44.



Looking at the Training and Validation loss, we can see both the Training Loss and Validation Loss have a large jump, it appears that the loss is leveling out and potentially gonna decline meaning if we trained the model with more Epochs, the loss could have potentially been lower.

Example with Crack

Inputted Image:  Result: The following image is predicted to have a crack: negative

Example without Crack

Inputted Image:  Result: The following image is predicted to have a crack: negative

Simple Convolutional Neural Network

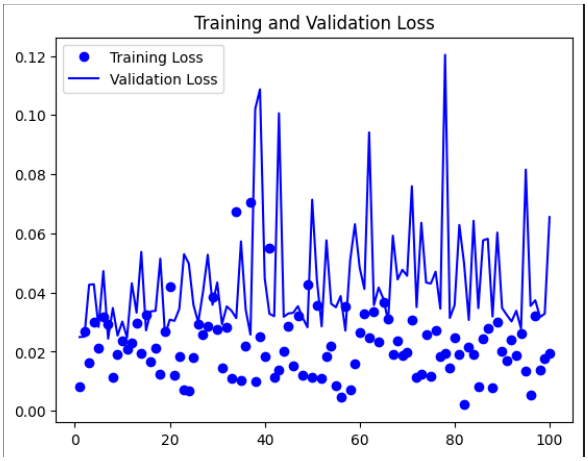
Recall	0.496
Precision	0.499
F1-Score	0.497
ODS	0.501
OIS	0.02475 (Invalid)
Accuracy	0.499

The model's Recall score was 0.496, which means the model can predict about half of the total positive samples as positive, meaning that the model has a similar number of True Positive samples as False Negative Samples. The model's Precision score was 0.499, which means about half of the samples the model predicts to be positive are positive, meaning the model has a similar number of True Positive samples as False Positive samples. The model's F1 Score was 0.497, and the model's Accuracy was 0.499. Both of these mean the model was only about 50%. The model's ODS was 0.501, meaning there is a better threshold for this model than 0.5. We could increase the ODS by changing the threshold value from 0.01 to 0.001 to find the best threshold and get the highest ODS possible, which would also get us the highest F1-Score. Changing the threshold to this new threshold would increase the model's Accuracy. The model's OIS was 0.02475; however, we believe this is invalid as it is a minimal value, and the OIS values they got in the paper ranged from 0.6 to 0.9. Overall, this model was not very good at predicting positive and negative samples, as shown by the low Recall, Precision, F1-Score, and Accuracy, and is only right about half of the time.

Positive Recall	0.50

Negative Recall	0.50
Positive Precision	0.50
Negative Precision	0.50
Positive F1-Score	0.50
Negative F1-Score	0.50

Because the model has the same Recall, Precision, and F1-Score, we can conclude that the model does not tend to favor positive or negative. It just is not entirely accurate at predicting positive or negative.



Looking at the Training and Validation Loss plot, we can see that the model is overfitting on the Training data, which can be shown by the Training Loss being lower than the Validation Loss. The overfitting is primarily consistent throughout the epochs; however, the peaks of the spikes do increase as the epochs increase, which further suggests the model is being overfitted on the Training data.

Example with Crack

Inputted Image:  Result: The following image is predicted to have a crack: negative

Example without Crack

Inputted Image:  Result: The following image is predicted to have a crack: positive

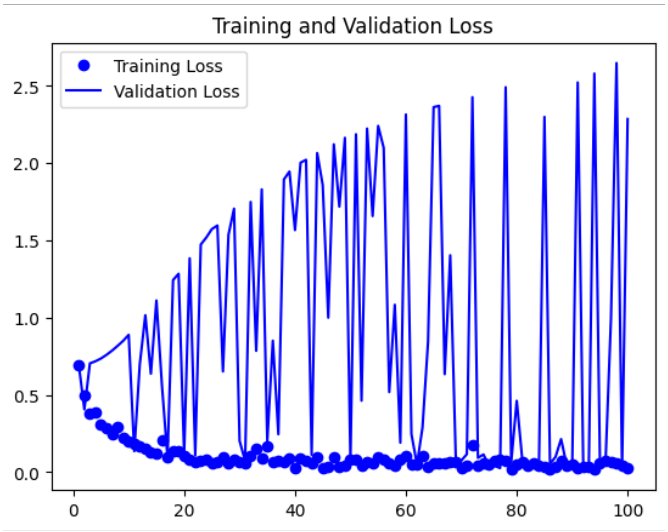
Complex Convolutional Neural Network

Recall	0.982
Precision	0.500
F1-Score	0.662
ODS	0.666
OIS	0.0245 (Invalid)
Accuracy	0.500

The model's Recall score was 0.982, which means the model can predict about half of the total positive samples as positive, meaning that the model has a similar number of True Positive samples as False Negative Samples. The model's Precision score was 0.500, which means about half of the samples the model predicts to be positive are positive, meaning the model has a similar number of True Positive samples as False Positive samples. The model's F1 Score was 0.662, and the model's Accuracy was 0.500. The moderate F1-Score suggests the model was moderately accurate at correctly predicting Positive samples. The Accuracy was measured using balanced Accuracy, based on the Sensitivity plus Specificity divided by 2. The model's ODS was 0.66, meaning there is a better threshold for this model than 0.5. We could increase the ODS by changing the threshold value from 0.01 to 0.001. The model's OIS was 0.0245; however, we believe this is invalid as it is a minimal value. Overall, this model was not very good at negative and moderately good at positive samples, as shown by the moderate Recall, Precision, F1-Score, and Accuracy. This can be further shown using the table below.

Positive Recall	0.98
Negative Recall	0.02
Positive Precision	0.50
Negative Precision	0.52
Positive F1-Score	0.66
Negative F1-Score	0.04

The model's Recall for Positive is almost perfect. However, its Precision for Negative is almost 0. This significant difference might be due to the model predicting Positive for most samples resulting in it being correct on Positive samples and incorrect on Negative samples. The model's F1-Score for Positive is also increasingly higher than its F1-Score for Negative, further supporting that theory.



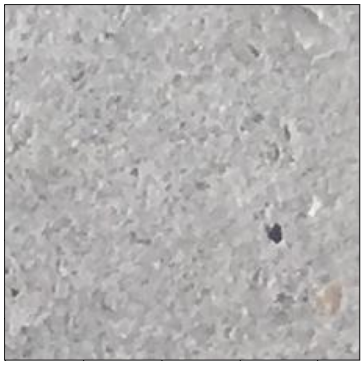
Looking at the Training and Validation Loss plot, we can see that the Training Loss and Validation Loss differ significantly. The Validation Loss is consistently higher than the Training Loss. This difference can be associated with the model overfitting on the Training Data and, as a result, causing the Validation Loss to be higher. The height

of the spikes also increases as the number of epochs increases, which further supports that the model is overfitting on the Training data.

Example with Crack

Inputted Image:  Result: The following image is predicted to have a crack: positive

Example without Crack

Inputted Image:  Result: The following image is predicted to have a crack: negative

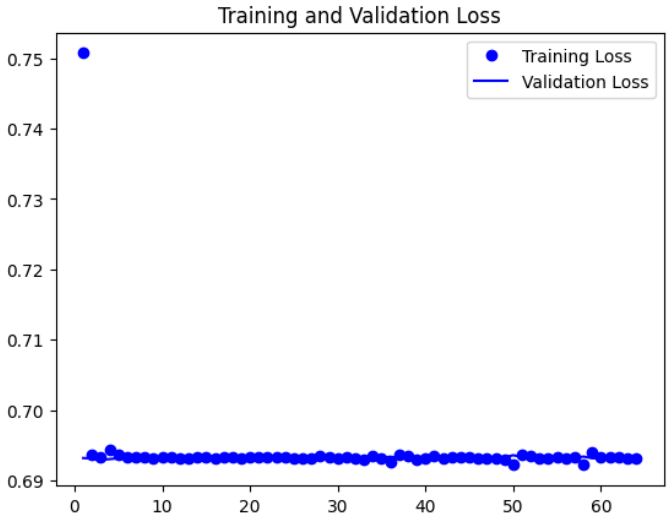
CNN with Encoder and Decoder

Recall	0.0
Precision	0.0
F1-Score	0.0
ODS	0.66
OIS	0.0 (Invalid)
Accuracy	0.5

As seen, the model's Recall and Precision are both 0. This is because the model only predicted Negative for all values, causing both values to be 0. Because F1-Score is determined by the Recall and Precision, it is also 0. The ODS was 0.66 meaning there is a better threshold for this model than 0.5. The model's OIS was 0.0; however, we believe this is invalid as it is a minimal value. Overall, this model was not very good at Positive and good at Negative samples, as shown by the low Recall, Precision, F1-Score, and Accuracy. This can be further shown using the table below.

Positive Recall	0.0
Negative Recall	1.0
Positive Precision	0.0
Negative Precision	0.5
Positive F1-Score	0.0
Negative F1-Score	0.67

The model could not correctly predict a Positive sample because it did not predict Positive for any samples. It only predicted Negative, which can be shown by the Negative Recall being 0. Because half the samples were Positive, but the model predicted them as Negative, the model's Negative predictions were only half right which can be shown by Negative Precision which is 0.5. The model's F1-Score for Positive is 0 because it did not predict a Positive sample, and its F1-Score for Negative is 0.67 and not 1.0 because the Negative Precision was 0.5.



Looking at the Training and Validation Loss plot, we can see that the Training Loss and Validation Loss are very similar, which makes it very difficult to see the Validation Loss because the Training Loss is almost on top of it for the entire plot.

Example with Crack

Inputted Image:  Result: The following image is predicted to have a crack: negative

Example without Crack

Inputted Image:  Result: The following image is predicted to have a crack: negative

Comparison

	U-HDN	Simple CNN	Complex CNN	CNN (enc-dec)

Recall	.396	0.496	0.982	0.0
Precision	0.496	0.499	0.500	0.0
F1-score	0.440	0.497	0.662	0.0
ODS	0.597	0.501	0.666	0.66
OIS	0.025 (Invalid)	0.02475 (Invalid)	0.0245 (Invalid)	0.0 (Invalid)
Accuracy	0.497	0.499	0.500	0.5

Looking at these results, it appears as if the Complex CNN is the best network at predicting Positive and Negative samples correctly. However, the Complex CNN had these high metric evaluations because it predicted Positive for almost every sample, including the Negative samples. These evaluation metrics evaluate the models based on their Positive predictions. If we look at the Negative predictions, the Complex CNN model is inferior at predicting. The CNN with Encoder and Decoder was the best model that correctly predicted negative samples. However, this was because this model predicted Negative for every sample resulting in its Negative Recall being 1.0 and Negative Precision being 0.50. The model that best predicted correctly for both Positive and Negative Samples was the U-HDN. The U-HDN had similar results for Positive and Negative for Recall, Precision, and F1-Score. The U-HDN was also trained with the fewest Epochs, with the Simple and Compex CNNS being trained with 100 Epochs, the CNN with Encoder and Decoder trained with 64 Epochs, and the U-HDN being trained with 16 Epochs. The U-HDN's performance could have been improved if we were able to train it with a larger amount of Epochs.

5 Potential Future Work

- Rechecking the U-HDN algorithm design we created and fixing any errors. Such errors may be from the multi-dilation module and the hierarchical feature learning module, where they may be placed or built in the wrong way.
- Examining other methods to deal with overfitting in models. This can include exploring different dropout thresholds that would be more effective in preventing overfitting.
- Checking for redundant feature maps and try to remove them from the network.
- Another potential task that can be done is retraining the U-HDN model and training it using a larger number of Epochs.
- Look for alternative ways to calculate the OIS to be more efficient on larger datasets.

6 References

[1] Fan, Z., Li, C., Chen, Y., Wei, J., Loprencipe, G., Chen, X., & Di Mascio, P. (2020). Automatic crack detection on road pavements using encoder-decoder architecture. Materials, 13(13), 2960. https://doi.org/10.3390/ma13132960

[2] Gad, A. F. (2021, April 9). *Accuracy, precision, and recall in deep learning*. Paperspace Blog. Retrieved April 15, 2023, from https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/

[3] Hardrock. (2021, April 8). Beware of floor cracks: The dangers concrete cracks can cause. Hard Rock Concrete Coatings | Utah Concrete Coating Contractor. Retrieved March 25, 2023, from https://www.hardrockconcretecoatings.com/beware-of-floor-cracks-the-dangers-concrete cracks-can-cause/#:~:text=Mold%2D%20if%20water%20is%20able,a%20source%20o %20potential%20injury.

[4] Özgenel, Çağlar Fırat (2019), "Concrete Crack Images for Classification", Mendeley Data, v2 http://dx.doi.org/10.17632/5y9wdsg2zt.2

[5] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

7 Task Delegation and Member Responsibilities

Nathan:

- Data Extraction: Downloading the data from Kaggle and then preparing for it to be split.
- Split Data into Training, Validation, and Testing subsets: Splitting the Data into 3 subsets: Training, Testing, and Validation. The Training subset contains 32,000 images, the Testing subset contains 4,000 images, and the Validation subset contains 4,000 images. Turn each subset into a folder to allow it to be shared. Zipped the Training folder and then uploaded the Training, Testing, and Validation folders to the shared google drive.
    - Added feature where pre-existing training, validation, and testing folders would be removed and re-added to build new datasets.
- Creating U-Hierarchical Dilated Network composed of three parts:
    - Unet with alterations based on the paper
        - Created 4 convolutional blocks in for the encoder and 3 convolutional blocks in the decoder.
        - Adding additional layers apart from the paper such as batch normalization and dropout.
    - Multi-Dilation Module
        - Created four convolutional blocks using different dilation rates with two convolutional layers each, and then combining them.
    - Hierarchical Feature learning module
        - Creating convolutional layers with different kernel sizes and different dilation rates before combining them all.

Nikolaus:

- Implementing alternative networks such as CNN with Encoder and Decoder, Simple Convolutional Neural Network, and Complex Convolutional Neural Network
    - Simple CNN: Created simple CNN containing 1 Convolutional Layer using the CNN code we were shown in class.
    - Complex CNN: Created a complex CNN containing multiple Convolutional Layers using the CNN code we were shown in class and the CNN code used for assignment 4.
    - CNN with Encoder and Decoder: Created a CNN similar to the Complex CNN however, implemented it so there was an Encoder and Decoder.
- Training the models: Compiled the models and then fit them using train_generator as the training data, steps per epoch being the length of the training data divided by 20, which is the batch size, 100 epochs, and validation_generator as the validation data. We then saved the model and the history to use later without retraining the model.
- Evaluating the models: Evaluated the models on several metrics: Recall, Precision, F1-Score, Accuracy, ODS, and OIS. We imported sklearn metrics to use its Recall_Score, Precision_Score, F1_Score, and Balanced_Accuracy functions. ODS is the best F1-Score for thresholds, so we use a for loop that got the F1-Score for each threshold from 0.01 to 0.99, incrementing by 0.01. We would then select the max F1-Score from the list of F1-Scores, which would be the model's ODS. OIS is the aggregate F1 score. It is similar to ODS; however, the Precision and Recall inside the equation have an exponent of i, which is the number of images. Because our testing data contains 4,000 images, we increment from 1 to 4,000. The testing to get the OIS was very intense; as a result, the PC we were using could not accurately get it, and the value returned was not valid.
- Comparing the different models' performances: Comparing the results of the evaluation metrics for each of the models and what they mean.