Problem Statement- Cole

Cracking in concrete can be the result of several factors, including stress from heavy loads, water intrusion, wear and tear, and shrinkage over time. Cracks can be a tripping hazard, scrap vehicles, or allow mold to form. Regular maintenance and proper monitoring can help prevent the formation of cracks in concrete. However, many places are not capable of being able to afford this regular maintenance and monitoring.

Background Materials- Cole

We got the dataset from Kaggle. The dataset contains 40,000 images split into 2 folders, Positive and Negative, with each folder containing 20,000. Note this dataset is different from the dataset used by the paper.

Proposed Solution- Nathan

The proposed solution is a U-Hierarchical Dilated Network, or U-HDN, which is based off of an encoder-decoder architecture for detecting cracks in pavement. We also created three additional convolutional neural networks to compare with the U-HDN. The first convolutional neural network was a simple cnn with a single convolutional layer, a complex cnn with 2 convolutional blocks of 2 convolution layers and batch normalization followed with max pooling, and a simple encoder-decoder cnn where the encoder and decoder sections both as two convolutional blocks similar to that in the complex cnn.

Implementation Overview- Nathan

For the U-HDN implementation, it is composed of 3 parts: a Unet architecture which is used to extract features and restore the original image, a multi-dilation module that is used to get more information of the features, and a hierarchical feature learning module which is used for obtaining multi-scale features. This network will train on 120 by 120 by 3 images of pavements with and without cracks and make predictions on whether inputted images have cracks.

UHDN Architecture- Nathan

In the image, we can see the UHDN architecture separated into three sections. The red dotted box represents the Unet architecture, the green dotted box represents the multi-dilation module, and the blue dotted box represents the HF learning module.

U-Net Architecture- Nathan

As said before, the Unet architecture is used to extract features at different spatial levels before reconstructing the original images. In both the encoder and decoder section, we have 4 convolution blocks, each having 2 3 by 3 convolutional layers and a 2 by 2 max pooling or upsamping after. We also decided to put batch normalization and dropout layers in between the layers. However, there are still problems with our implementation and still need to experiment

with the regularization impacts and parameters. To fit the image dataset, we removed the 4th pooling layer, 9th and 10th convolution layers, and the first up-convolution layer. This was done to avoid reducing the image from 15 by 15 to 7 by 7, which will cause an equality in image shapes where upsampling will result in a 14 by 14 image.

Multi-Dilation Module (MDM) Architecture- Nathan

In between the encoder and decoder sections, we implement the multi-dilation module, which functions in capturing features at multiple spatial scales. As followed in the image, we take the output of the encoder and pass them into four separate convolutional blocks, each with 2 3 by 3 convolutional layers. For each of the blocks, we apply a different dilation rate of 2, 4, 8, and 16. We then take each of these results and concatenate them and apply a 1 by 1 convolutional layer to them. The paper did not specify how many filters were applied to each convolutional layer, so we decided on 64 filters for each layer. As such, these values can be taken into account for future alterations and improvements.

Hierarchical Feature (HF)- Nathan

In the last part of the UHDN, we have the HF learning module, which aims to extract high level feature maps at the high and low convolutional levels. In our case, we implemented it with three convolutional layers with different kernel sizes of 1 by 1, 3 by 3, and 5 by 5. We also applied three convolutional layers of different dilation rates of 2 4 and 8. We then concatenated all these and outputed them to the following layers. At this step, we see many problems and improvements that can be made, such as removing repeated features. The overall number of parameters the model had was 19 million. In order to improve the model, we would look into reducing the repeated features to simplify the network and reduce the number of redundant parameters while retaining its accuracy. We also ran into another issue of logit and label shapes not matching in the input and output layers, which resulted in us creating a makeshift solution of flattening and dense layers. Of course, in future works, we would need to further analyze the model structure to solve the mismatch in shapes.

U-HDN Results- Cole

The Evaluation Metrics we used are Recall, Precision, F1-Score, ODS, and OIS. ODS is the best F1 on the public database for a fixed threshold. OIS is the aggregate F1 on the public database for the best threshold in each image. We believe that the OIS scores we got for all 4 models are invalid. This is because the function we created to calculate the OIS utilizes a double for loop with one loop going from i to the number of images, which for the test subset is 4,000 images, and the other loop going through the Threshold (0.01 to 0.99). We got a bunch of Scalar errors during this process before getting a value that is very low for ODS. We believe this invalid value is the result of the function being very computationally intense and the computer we were running it on not being capable of running it correctly. As we can see by the Training and Validation Loss plot, for the number of Epochs we used, the model fits the Validation Data pretty

well. Given more time, we would have increased the number of Epochs we used to train, as this would increase the number of points on the plot, which would have given us a better understanding of how the model fits the Validation Data. The model still predicts Positive and Negative samples correctly about 50% of the time. We can see this by looking at the model's Recall, Precision, and F1-Score for both Positive and Negative. For all of these metrics, we see a score of nearly 50%. We believe the model's accuracy would have been improved had we increased the number of Epochs we used to train. We could also improve the accuracy of the model by changing the Threshold. ODS is calculated by getting the F1-Score for every Threshold ranging from 0.01 to 0.99. Because the ODS is higher than the F1-Score, this means that there is a better Threshold than 0.5. If we choose to use this Threshold instead of 0.5, the model's accuracy will increase.

U-HDN Demonstration- Cole

As we can see, the model falsely predicts Positive samples as Negative and Negative samples as Positive, but it still correctly predicts Positive and Negative samples 50% of the time.

Simple CNN Results- Cole

As we can see by the Training and Validation Loss plot, the model overfits the Training Data resulting in higher Validation Loss than Training Loss. The model still predicts Positive and Negative samples correctly about 50% of the time. We can see this because, for both Positive and Negative, the model's Recall, Precision, and F1-Score are 0.50. This model's accuracy can be increased by changing the Threshold from 0.5 to the Threshold when the ODS is 0.501.

Simple CNN Demonstration- Cole

As we can see, the model falsely predicts Positive samples as Negative and Negative samples as Positive, but it still correctly predicts Positive and Negative samples 50% of the time.

Complex CNN Results- Cole

Cole: As we can see by the Training and Validation Loss plot, the model overfits the Training Data resulting in higher Validation Loss than Training Loss. Looking at the Recall, the model correctly predicts Positive almost 100% of the time, while it correctly predicts Negative almost 0% of the time. This can be understood by looking at the model predicting Positive and Negative sample images. This model's accuracy can be increased by changing the Threshold from 0.5 to the Threshold when the ODS is 0.666.

Complex CNN Demonstration- Cole

The Model predicts Positive almost 100% of the time, which is why the Model's Recall is 0.982, but its Precision is only 0.50

CNN with Encoder and Decoder Results- Cole

As we can see by the Training and Validation Loss plot, the model fits the Validation Data pretty well, resulting in the Training and Validation Loss being basically the same. Looking at the Recall, the model correctly predicts Negative 100% of the time while it correctly predicts Positive 0% of the time. This can be understood by looking at the model predicting Positive and Negative sample images. This model's accuracy can be increased by changing the Threshold from 0.5 to the Threshold when the ODS is 0.66.

CNN with Encoder and Decoder Demonstration- Cole

The Model predicts Negative 100% of the time, which is why the Model's Negative Recall is 1.0, but its Negative Precision is only 0.50

Results Comparison- Cole

Cole: The Complex CNN model was best at predicting Positive samples correctly however, this is due it almost only predicting Positive. The CNN with Encoder and Decoder was best at predicting Negative samples correctly however, this is due to it only predicting Negative. Both the UHDN and Simple CNN predicted Positive and Negative correctly around 50% of the time. We believe that although the UHDN and Simple CNN had similar results, the UHDN is still the better choice because it was trained on 16 Epochs, whereas the Simple CNN was trained on 100 Epochs. Had we trained the UHDN with more Epochs, we believe that the accuracy of the model would have increased. Had we chosen the optimal Threshold for every model, the Complex CNN would have had the highest accuracy, followed closely by the CNN with Encoder and Decoder, then the UHDN, and finally, the Simple CNN. We could improve the accuracy of all the models. While the Complex CNN and CNN with Encoder and Decoder both have higher ODS scores, we believe this is the result of the UHDN being trained with far fewer Epochs than any other model. Had we trained the UHDN with more Epochs, then we believe that the UHDN would have a higher ODS than any other model, which would mean that the UHDN would be the most accurate of the models.

Additional Notes

<u>UHDN Architecture- Nathan</u>

Blank

<u>How the Results Work- Cole</u>

- 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.

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

- Accuracy: ratio of the number of correct predictions to the total number of predictions.

Problem Notes:

- Preprocessing
- Batch normalization
- Output layer (we used flatten and dense layer)
- HF learning block
- Need to run more epochs
- Activation functions
- Filter numbers in convolutional layers
- Should have taken a subset of the dataset to test the models (reducing runtime)
- Dilation rates can be experimented