COE379L Project 3 Report

Ayushi Sapru

as98489

To prepare the dataset, I loaded a set of satellite images labeled as either “damage” or “no_damage” from Hurricane Harvey’s impact. The data was split into training, validation, and test sets using TensorFlow’s ‘image_dataset_from_directory.’ I standardized all images to a fixed size of 128x128 pixels and normalized the pixel values to fall between 0 and 1 to improve model performance. I performed basic checks such as confirming the number of images per class and verifying the shape and data types of the image arrays.

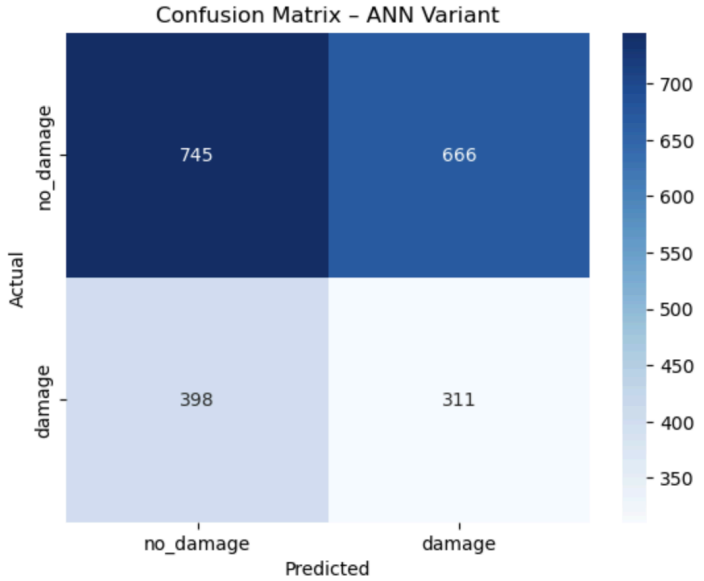
I explored 3 different model architectures: ANN (Artificial Neural Network), LeNet-5 CNN (Convolution Neural Network), and Alternate LeNet-5. My initial ANN flattened each image and passed the data through a few dense layers. The model consistently predicted "no_damage" regardless of input, resulting in poor performance. I tried to address this by adding dropout to reduce overfitting and increasing the number of neurons in each layer. With that, ANN was able to recognize both classes and became more balanced, but still struggled to identify “damage” correctly. For LeNet-5 CNN, I implemented a modified version of the original - this model included convolution layers followed by average pooling, flattening, and fully connected layers. LeNet-5 dramatically outperformed ANN and achieved better classification accuracy. I also tried a variant of LeNet-5 based on the research paper provided, which included more filters and deeper dense layers. However, it underperformed compared to the standard LeNet-5 in my use case. It showed high error rates and inconsistent predictions. Ultimately, I chose the standard LeNet-5 CNN as the final model because it demonstrated the best performance and generalization capability.

The LeNet-5 CNN model achieved 93% test accuracy and maintained high precision and recall, particularly for the “damage” class. This was important because my earlier models had

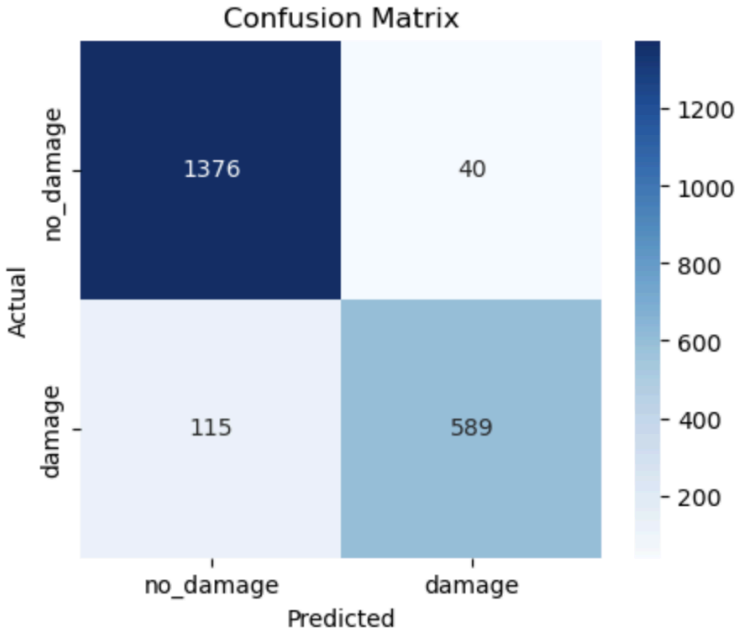
trouble recognizing the “damage” class entirely, and counted 0 until I retrained it. I validated the model’s performance using confusion matrices and classification reports. I’m relatively confident in this model’s ability to generalize new images, however, further enhancement could help the model’s performance. As I was testing against the given test cases, I was not able to get more than 50% accuracy, even after enhancing the model.

For model deployment, I used a Flask inference server. The server has two endpoints: GET /summary, which returns model metadata - like architecture and layer details - in JSON format, and POST /inference, which takes an image in binary format and returns a classification result as JSON. In our case, the prediction is either “damage” or “no_damage.”

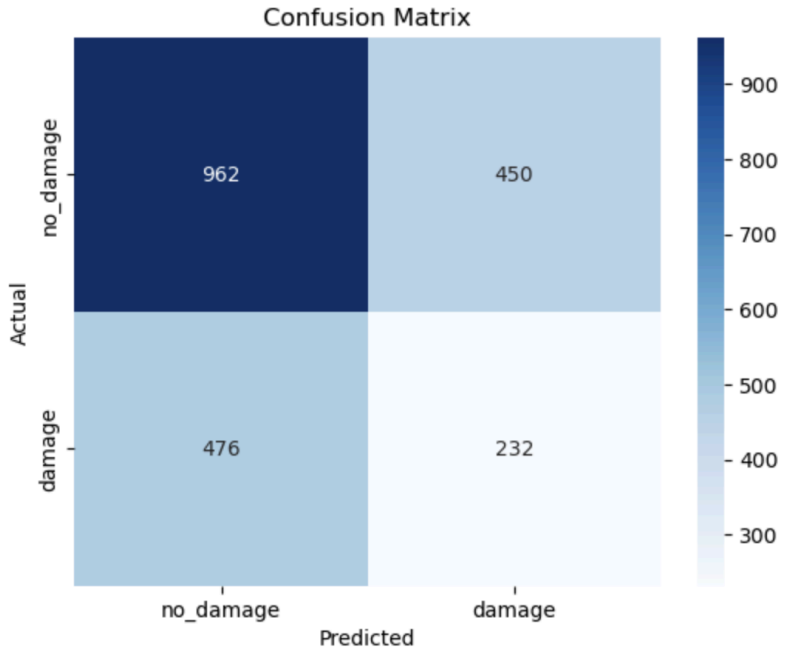
I containerized the entire server using Docker and pushed the image to Docker Hub under ayushisapru02/inference_server. I also created a docker-compose.yml file to simplify container creation. I’ve added instructions in the README.md file to guarantee easy deployment. Users can send a POST request to the /inference endpoint with an image file, and the server will resize and normalize the input image automatically. The response is a simple JSON script like {"prediction": "damage"}, and this makes it easy for other applications to interact with the model.



Confusion Matrix for ANN Variant



Confusion Matrix for LeNet-5 CNN



Confusion Matrix for Alternate LeNet-5 CNN