Project 3 Report – Hurricane Harvey Damage Classification

1. Data Preparation

The first stage of this project focused on preparing the dataset for machine learning. The dataset consisted of real satellite images taken after Hurricane Harvey, stored in two folders: one containing images labeled as “damage” and the other labeled as “no_damage.” Before constructing any models, I began by loading the images, examining the dataset structure, and reviewing several sample images to understand what visual cues might signal building damage. This initial inspection revealed noticeable patterns such as roof collapse, debris fields, and structural distortion in damaged images, as well as the challenges posed by variations in lighting, angles, and surrounding terrain.

To ensure training stability and consistency across models, all images were converted to RGB format and resized to 128×128 pixels. This decision standardized the input dimensions and simplified the modeling process. Pixel values were normalized to the [0,1] range, which improves gradient-based optimization and prevents numerical instability during training. After preprocessing, the dataset was split into a 70% training set, 15% validation set, and 15% test set, with balanced class representation in each split. This structure ensured that models would be evaluated fairly on unseen data and prevented leakage during training. By the end of preprocessing, I had a fully normalized, uniformly sized collection of images suitable for deep learning.

2. Model Design and Architectural Decisions

The second stage of the project involved exploring multiple neural network architectures to determine which would perform best for the classification task. I implemented and evaluated three models in accordance with the project requirements: a fully connected artificial neural network (ANN), a classical LeNet-5 convolutional neural network, and an Alternate-LeNet-5 architecture inspired by the referenced research paper. Each model introduced different design considerations and provided insights into how architecture influences performance on image classification tasks.

2.1 Fully Connected ANN

The ANN served as the simplest baseline model. It flattened each 128×128×3 image into a 49,152-element vector, passed it through several dense layers with ReLU activation, and produced a binary prediction through a sigmoid output layer. Although easy to implement, this model suffered from a fundamental weakness: flattening the image removes spatial

relationships between pixels. Because identifying damage relies heavily on patterns distributed across the image—structural outlines, texture changes, shading anomalies—the ANN lacked the representational capacity needed to learn these patterns effectively. During training, it showed signs of overfitting, high validation loss, and poor generalization, making it unsuitable for this problem.

2.2 LeNet-5 CNN

To improve performance, I implemented a modernized version of the classical LeNet-5 architecture. CNNs maintain spatial structure through convolutional filters, making them far better suited for image tasks. The model included convolutional layers with ReLU activation, max-pooling layers for reducing dimensionality, and fully connected layers leading to the final classification decision. This architecture captured patterns such as edges, shapes, and texture differences that the ANN could not learn, significantly improving accuracy and stability. However, LeNet-5 still had limitations because its relatively shallow architecture struggled to capture more subtle forms of structural damage present in certain satellite images.

2.3 Alternate-LeNet-5 CNN

The most effective architecture was the Alternate-LeNet-5 model, which incorporated deeper convolutional layers with higher filter counts (32, 64, and 128 filters) and two fully connected layers with 512 and 256 neurons, respectively. Dropout layers were included to reduce overfitting and force the network to learn more robust patterns. This architecture's increased depth and feature extraction capacity made it particularly effective at identifying fine-grained visual cues such as partial roof loss, water pooling, debris distribution, and shadowing irregularities around damaged structures. The Alternate-LeNet-5 architecture ultimately offered the best balance between model complexity and generalization performance.

3. Model Evaluation

To compare the models objectively, I trained them under identical conditions—same input shape, same train/validation/test splits, and similar optimization settings—and evaluated their performance using accuracy and loss metrics on the test dataset. As expected, the ANN performed the worst, demonstrating significant overfitting and unstable validation accuracy. LeNet-5 performed considerably better due to its ability to learn spatial patterns; however, it still struggled with nuanced cases.

The Alternate-LeNet-5 model achieved the highest performance, with a test accuracy of approximately 86.7%. It exhibited smooth training and validation curves, minimal overfitting, and strong generalization to unseen images. Beyond numerical metrics, I evaluated the model qualitatively by manually testing images from outside the dataset. The model consistently produced correct predictions for both obvious and subtle examples of building damage. The strongest validation came from running the official course grader against my deployed inference server: the grader evaluated six images and confirmed correct predictions for all of them,

yielding 6/6 accuracy. This demonstrated not only strong model performance but also full API compliance. Based on these results, I have high confidence that the Alternate-LeNet-5 model is reliable and effective for this classification task.

4. Model Deployment and Inference Server

The final stage of the project required deploying the trained model as an inference server capable of responding to HTTP requests. I implemented the server using Flask, building two endpoints consistent with the project's API specification. The first endpoint, `/summary`, returns a JSON object describing the model's architecture, preprocessing requirements, input shape, and parameter count. The second endpoint, `/inference`, accepts raw binary image data, processes it internally, and returns a JSON response containing a single field: "prediction," with one of the two required values—"damage" or "no_damage." This format is essential for compatibility with the automated grader.

For deployment, I created a Dockerfile that installs TensorFlow, Flask, Pillow, NumPy, and other dependencies in a lightweight Python 3.11-slim environment. I built the container on the course VM to guarantee x86 architecture compliance, as required. I also prepared a docker-compose configuration to simplify management of the inference server. Running the server requires only "docker compose up," while shutting it down requires "docker compose down." Once running, the server exposes port 5000 and can be tested using standard curl commands. For example, "curl http://localhost:5000/summary" retrieves metadata, while "curl -X POST --data-binary @image.jpeg http://localhost:5000/inference" returns a prediction for a real image. I included these examples in the README as instructed.

To validate deployment correctness, I executed the provided grader script, which programmatically tested both endpoints. The grader confirmed that my server returned correctly formatted JSON and accurate predictions on all test images. This final step verified that the deployment was functional, compliant, and fully integrated.

Conclusion

This project covered the full machine learning workflow—from data preparation to model deployment. By carefully analyzing the dataset, designing and comparing multiple architectures, selecting the strongest model, and deploying a fully functional inference server, I developed a complete system capable of classifying building damage in satellite imagery. The Alternate-LeNet-5 architecture proved most effective, and its performance was validated both quantitatively and via the automated grader. The successful Docker deployment ensures the model can be reproduced, evaluated, and executed consistently across different environments. This project demonstrates how deep learning, when combined with rigorous evaluation and proper deployment, can contribute meaningfully to real-world disaster analysis and response efforts.