Project 3 Report – Hurricane Harvey Building Damage Classification

1. Data Preparation (1 pt)

The dataset contained satellite imagery from Texas following Hurricane Harvey. Images were already organized into two folders: `damage/` and `no_damage/`, corresponding to whether the building in the image had visible storm damage.

My goals in data preparation were:

1. Load and label the images,
2. Explore their dimensions, quality, and consistency,
3. Prepare them for training across multiple neural network architectures.

Image Loading and Inspection

I loaded all images using TensorFlow and PIL, verified their count, and confirmed that the dataset was balanced closely enough for binary classification. I examined a sample of images from each category to understand visual patterns—damaged buildings typically had debris, roofs missing, or visible structural deformations, while non-damaged structures were intact.

Resizing and Scaling

Since the raw images varied in resolution, I resized all images to **128×128 RGB** to standardize input shape across all models.
Pixel values were normalized to the range **[0, 1]** for stable gradient descent during training.

Dataset Splitting

I divided the processed images into:

- **70% training**
- **15% validation**

- **15% testing**

This ensured enough data for training while preserving a clean, unseen test set for final evaluation.

---

2. Model Design and Architecture Exploration (2 pts)

To follow the assignment requirements, I implemented and compared **three families of neural network architectures**:

A. Fully Connected ANN

The ANN served as a baseline model.
Design choices:

- Flatten input image (128×128×3 → 49,152 features)
- Hidden layers: ReLU activations
- Output layer: sigmoid for binary classification

Observations:
This architecture performed the worst due to the loss of spatial information. Dense networks do not handle images efficiently because they treat pixels as independent features.

B. Classic LeNet-5 CNN

I implemented a modified LeNet-5 adapted for RGB 128×128 images:

- Two convolutional layers with ReLU
- Max-pooling layers for spatial reduction
- Fully connected layers followed by a sigmoid classifier

Why this model performed better:
CNNs extract spatial patterns—edges, textures, structural cues—which are critical for analyzing aerial images of damage.

C. Alternate-LeNet-5 (from the assigned research paper)

This was the third and ultimately the **best-performing architecture**.

Design decisions:

- Increased depth: 32 → 64 → 128 filters in successive layers
- Larger feature maps and more expressive filters
- Two dropout layers to reduce overfitting
- Fully connected layers of sizes 512 and 256

Why this was effective:
The deeper convolutional layers captured fine-grained structural features like roof tears, debris patterns, and shading differences between damaged and intact buildings.

---

3. Model Evaluation (1 pt)

I trained all three architectures under the same conditions—same image size, batch size, train/val/test splits, and number of epochs. I evaluated performance using accuracy, loss curves, and validation metrics.

Performance Summary

- **ANN:** lowest accuracy, heavily overfit, poor generalization
- **LeNet-5:** significantly better, learned spatial features well
- **Alternate-LeNet-5:** highest accuracy and lowest loss

Best Model

The **Alternate-LeNet-5** model achieved the best overall results with:

- **Test accuracy: ~86.7%**
- Smooth training/validation curves
- Consistently strong performance on new images

I am confident in this model because:

- It generalizes well to unseen test data
- It performed perfectly on the provided inference grader (6/6 correct predictions)
- Its architecture is specifically suited to image tasks involving structural features

While satellite damage detection still has inherent ambiguity, the model is reliable for this dataset and task.

---

4. Model Deployment and Inference (1 pt)

The final step was to deploy the trained model as an inference server using **Flask + Docker**, as required.

Deployment Steps

1. Saved the trained model as `best_model.h5` and `best_hurricane_model.keras`.
2. Implemented a Flask server (`app.py`) with two endpoints:
   - **GET `/summary`** → returns metadata such as input size, architecture name, preprocessing steps

POST `/inference` → accepts raw image bytes and outputs

{ "prediction": "damage" }
or

{ "prediction": "no_damage" }

-

3. Packaged the server in a Docker image using an x86-compatible Dockerfile built on the course VM.

4. Ran the server with `docker-compose.yml`.

Running the Inference Server

To start the server:

docker compose up

To stop the server:

docker compose down

Example Inference Requests

Retrieve model summary:

curl http://localhost:5000/summary

Perform inference:

```
curl -X POST \
  --data-binary "@data/damage/example.jpeg" \
  http://localhost:5000/inference
```

Grader Verification

Using the provided grader script:

bash start_grader.sh

Result:

Total correct: 6
Accuracy: 1.0

This confirms the server exactly matches all assignment requirements.

✔️ Conclusion

Across the four parts of the project:

- I cleaned, explored, and preprocessed the Hurricane Harvey dataset.
- I implemented three neural architectures and selected the best model based on performance.
- I deployed my best model in a fully Dockerized inference server that accepts binary image input.
- My implementation passed the official grader with perfect accuracy and conforms to all API specifications.

This concludes Part 4 of the project.