1. Data Preparation

The dataset consists of satellite images taken after Hurricane Harvey, labeled as **damage** or **no_damage**. All images were provided as 128×128 RGB JPEGs. During loading, each file was decoded, converted to RGB, and normalized by dividing pixel values by 255. Normalization ensured stable gradients and faster convergence across the CNN architectures explored later.

After loading, the dataset contained approximately **14,000+ damaged** and **7,000+ non-damaged** buildings—an inherent **class imbalance**. To address this, I computed class weights based on the inverse frequency of each class and applied them during training. This prevented the model from defaulting to the majority "damage" prediction and encouraged balanced learning.

To maintain strict dataset separation, I implemented a **stratified 70/15/15 split**. A custom splitting function ensured that each class maintained its proportions across the training, validation, and testing sets. The final splits contained:

- **Train:** 1,073 images; **Validation:** 268 images; **Test:** 268 images

All datasets were wrapped into performant tf.data.Dataset pipelines using batching (batch size = 32), shuffling, prefetch, and parallel loading via AUTOTUNE. This kept GPU/CPU utilization high and reduced I/O bottlenecks during training.

Before modeling, I performed exploratory visualization of random samples. Visual distinctions such as roof discoloration, debris fields, exposed interiors, and missing shingles were clear in the damaged class. Non-damaged homes displayed intact roof geometry and fewer structural artifacts. These qualitative differences supported the feasibility of CNN-based classification.

2. Model Design

I implemented and trained three architectures as required by the assignment:

2.1 Fully Connected Dense Network

This baseline model flattened the 128×128×3 images into 49,152-element vectors. Despite adding several dense layers with ReLU activation, dropout, and L2 regularization, performance was limited due to loss of spatial information, high parameter count, and sensitivity to noise. Validation accuracy plateaued at **~91%**, making it unsuitable as the final model.

2.2 LeNet-5 Convolutional Neural Network

I reimplemented the classical LeNet-5 architecture with minor adaptations for 128×128 RGB input:

- Three convolutional layers (32 → 64 → 120 filters)
- Max-pooling after each convolution
- Fully connected layers (84 → 1)
- Final sigmoid activation for binary classification

This model achieved:

- **Val Accuracy:** ~95%, **stable training curves, and strong generalization** with no overfitting. Its simplicity and clean performance made it a strong candidate model.

2.3 Alternate-LeNet (from research paper)

This was the most advanced model in the project. Compared to standard LeNet:

- Higher channel counts (32 → 64 → 128)
- Larger fully connected layers (256 → 64 → 1)
- Dropout for regularization

This architecture preserved the spatial hierarchies of roofs and shadows and learned richer texture representations. It produced the best results of all experiments.

3. Model Evaluation

After training all three models under identical preprocessing and splits, their final validation accuracies were:

Model	Validation Accuracy
Dense	**0.91**
LeNet-5	**0.96**
Alternate-LeNet	**0.99**

The **Alternate-LeNet model** clearly outperformed the others.

On the **held-out test set**, it achieved:

- **Accuracy:** 99%
- **Precision:** 0.99
- **Recall:** 1.00 for “damage”
- **ROC-AUC:** 0.9998
- **Confusion Matrix:**
    - True Damage: 206 correct

True No-Damage: 60 correct
Only **2 total misclassifications**

These results suggest extremely high performance and a well-calibrated model. Because the test set was stratified and separated cleanly, I have high confidence in the model's generalization. Thus, the **Alternate-LeNet model** was selected for deployment.

4. Model Deployment & Inference Server

The best model was exported using: model.save("saved_models/best_model.keras"). I built a simple Flask inference server with two required endpoints:

GET /summary

Returns: input shape, number of parameters, model name, and description.

POST /inference

- Accepts **raw binary JPEG bytes** (no base64).
- Automatically resizes to 128×128, normalizes, and runs inference.
- Returns JSON with "prediction": "damage" or "no_damage".

The server was fully containerized using:

- Dockerfile for reproducible build
- docker-compose.yml to expose port 5000
- Image pushed to Docker Hub per assignment requirements

This allows instructors to grade using the automated HTTP-based test harness.

5. Conclusion

This project demonstrated the full pipeline of computer vision modeling:

- Preprocessing imbalanced real-world satellite imagery
- Designing and comparing neural models
- Selecting the best architecture using rigorous evaluation
- Deploying a real inference service in Docker

The alternate-LeNet architecture achieved excellent performance and deployed cleanly. The workflow highlights how classical CNN designs, when tuned appropriately, remain highly effective for binary classification on small-resolution imagery.