Project 2 Report: Damage Detection Model and Inference Server

1. Data Preparation

The project analyzed aerial post-disaster images which were separated into two distinct groups: damage and no_damage. The dataset consisted of JPEG images which presented different dimensions and positions and illumination settings.

Data preprocessing was handled entirely in TensorFlow/Keras. The images underwent a process of resizing to 128×128 pixels followed by RGB conversion and normalization to the [0, 1] range for achieving uniform input values. The training data received 70% of the total distribution while validation and testing received 15% each to achieve proper training exposure and evaluation accuracy.

The model reached stable convergence because of the combination of batch normalization with shuffling and normalization layers. The researchers first applied data augmentation through flips and rotations but later eliminated it to achieve deterministic and reproducible results for automated grading.

The design created an efficient lightweight preprocessing system that runs inside Docker while delivering consistent results on all hardware platforms and devices.

2. Model Design

The research examined three CNN models starting with a basic model followed by increasingly complex yet operational architectures. The objective focused on achieving an optimal combination between model complexity and accuracy and fast inference performance.

2.1 Baseline Model

The baseline CNN included:

- Two convolutional layers (3×3 filters, ReLU activation)
- One max-pooling layer
- One dense sigmoid output neuron

Although this model reached ≈80% validation accuracy, it underfit complex features, revealing the need for deeper feature extraction.

2.2 LeNet-Inspired Model

A second model adopted the LeNet-5 structure:

- Three convolutional blocks using ReLU activation and max pooling
- Flatten and dense layers for dimensionality reduction
- Dropout for regularization
- A final sigmoid output neuron for binary classification

This model improved validation accuracy to ≈88–90%, demonstrating more robust learning while avoiding overfitting.

2.3 Final Model: alt_lenet5_paper

The best-performing architecture was a refined LeNet variant named alt_lenet5_paper. The structure included:

- Conv(32) → Conv(64) → Conv(128) blocks with max pooling
- Dropout(0.3) after dense layers for regularization
- Adam optimizer with a learning rate of 1e-3
- Binary cross-entropy loss function

This model contained ≈1.2 M parameters—small enough for fast inference but deep enough for rich feature extraction. It became the final choice for deployment due to its excellent generalization and compatibility with real-time inference.

3. Model Evaluation

All models were trained for up to 20 epochs with early stopping to prevent overfitting. Performance was monitored through TensorBoard.

Metric	**Baseline**	**LeNet**	**alt_lenet_paper**

Train Accuracy	74%	93%	94%
Val Accuracy	80%	89%	91%
Test Accuracy	78%	88%	90%

The final model achieved strong agreement between validation and test results, confirming high generalization. Most misclassifications occurred in borderline cases (e.g., subtle or shaded damage).

The model was exported as artifacts/best_model.keras and paired with model_summary.json for metadata storage, allowing it to reload seamlessly in the Flask server.

4. Model Deployment and Inference

The final model was served through a lightweight Flask inference API that met the automated grader's specifications. The server provided two endpoints:

4.1 Inference Server Overview

GET /summary

Returns model metadata in JSON format.

Request

```
curl http://localhost:5000/summary
```

Response

```
{
 "best_model_name": "alt_lenet5_paper",
"img_size": [128, 128],
  "channels": 3,
 "class_names": ["damage", "no_damage"]
}
```

POST /inference

Accepts raw image bytes and returns a prediction.

Request

```
curl -X POST -H "Content-Type: application/octet-stream" \
  --data-binary "@/path/to/image.jpg" \
 http://localhost:5000/inference
```

Response

```
{"prediction": "damage"}
```

Both endpoints were tested on the student VM and returned correct, grader-compliant JSON. The model correctly classified images from both categories.

4.2 Docker Deployment

The inference server was containerized for reproducibility and platform independence. The Dockerfile used python:3.10-slim as its base image, installed dependencies from requirements.txt, copied the application files to /app, exposed port 5000, and executed server.py on startup.

Example build and run commands:

```
docker build -t project02-inference .
docker run -p 5000:5000 project02-inference
```

A docker-compose.yml file was included to simplify container management.

Example usage:

```
docker-compose up -d
docker-compose down
```

The container was built and tested on the UT class VM (x86) to ensure compatibility with the course grader.
The Flask app served inference requests through port 5000 successfully.

4.3 Usage Summary

Steps to run and test the model:

1. Build the Docker image or use Docker Compose.
2. Start the container.
3. Use curl or the provided grader script to send:
    - GET /summary → Retrieve model metadata
    - POST /inference → Classify an image
4. Stop the container after testing.

Example end-to-end workflow:

```
docker-compose up -d
curl http://localhost:5000/summary
curl -X POST -H "Content-Type: application/octet-stream" \
    --data-binary "@test_image.jpg" \
    http://localhost:5000/inference
docker-compose down
```

5. Conclusion

This project demonstrated a full machine-learning-to-deployment pipeline:

1. Data Preparation — Standardized, normalized, and reproducible image preprocessing.
2. Model Design — Iterative CNN experimentation leading to a compact, high-accuracy alt_lenet5_paper model.
3. Evaluation — The model achieved a test accuracy of 90% while showing no signs of significant overfitting problems.
4. Deployment — The system includes a deployment mechanism which enables HTTP-based access to the inference server through Docker containerization.

The pipeline functions as a dependable transportable system which enables future ML engineering and MLOps operations.