Moksh Nirvaan
COE 379L

1. Data Preparation

The dataset consisted of satellite images from Texas after Hurricane Harvey, categorized into two classes: damage and no_damage. The dataset had 14,170 images labeled as damage and 7,152 as no_damage, totaling 21,322 images.

The following preprocessing and data preparation steps were applied:

- **Image Resizing:** All images were resized to a fixed resolution of 128x128 to ensure consistent input dimensions.
- **Normalization:** Images were normalized using a mean and standard deviation of 0.5 across all channels to center the data.
- **Data Splitting:** The dataset was split into training (80%), validation (10%), and test sets (10%) to evaluate model performance fairly.
- **Transforms:** Used `torchvision.transforms` to apply `Resize`, `ToTensor`, and `Normalize` operations.

Data was loaded using `torchvision.datasets.ImageFolder` and wrapped in `DataLoader` objects for efficient training with batch processing.

2. Model Design and Development

Three different model architectures were explored:

a. Fully Connected Dense ANN

- Architecture: 128x128x3 image flattened to 1D, followed by two dense layers (1024 → 512 → 2 output).
- Activation: ReLU, Output: Softmax via `CrossEntropyLoss`
- Performance was limited due to the model's inability to capture spatial information from images.

Moksh Nirvaan
COE 379L

b. LeNet-5 CNN (Original)

- Architecture:
    - Conv2d(3, 6, kernel_size=5) → MaxPool → Conv2d(6, 16, kernel_size=5) → MaxPool
    - Flatten → FC(256 → 120) → FC(120 → 84) → FC(84 → 2)
- Showed decent performance with basic spatial feature extraction.
- **Validation Accuracy:** ~78.4%
- **Test Accuracy: 81.5%**
- Overfitting is observed after a few epochs, despite stable training.

c. Alternate LeNet-5 CNN (Best Performing)

Inspired by [https://arxiv.org/pdf/1807.01688.pdf], the alternate architecture included:

- Padding added in convolutional layers to preserve spatial dimensions
- Dropout layer after first FC layer to reduce overfitting
- Same input dimension support (128x128x3)

Final Architecture:
Conv2d(3, 6, kernel_size=5, padding=2) → ReLU → MaxPool
Conv2d(6, 16, kernel_size=5, padding=2) → ReLU → MaxPool
Flatten → FC(16*32*32 → 120) → Dropout(0.5) → FC(120 → 84) → FC(84 → 2)

Several training strategies were explored with this model:

- **Early Stopping:** Triggered at epoch 11 with **Test Accuracy = 94.98%**
- **Full 30 Epoch Training:** Achieved best generalization with:
    - **Validation Accuracy: 96.7%**

- **Final Test Accuracy: 96.7%**

3. Model Evaluation (1 pt)

Model	Final Validation Accuracy	Test Accuracy
Dense ANN	~75%	~76%
LeNet-5 (Original)	78.4%	81.5%
Alternate LeNet-5 (ES)	93.4%	**94.98%**
Alternate LeNet-5 (30)	**96.7%**	**96.7%**

Best Model: Alternate LeNet-5 (30 epochs)

- This model had the best accuracy and generalization, with smooth convergence and high confidence.
- Dropout and architectural tuning (padding, deeper layers) helped improve generalization.
- Overfitting was well controlled even at 30 epochs.

Evaluation was done using training logs, accuracy scores, and qualitative review of predictions.

4. Model Deployment and Inference

The best model was exported using:

torch.save(model.state_dict(), "optimized_lenet5.pth")

Deployment Setup:

- A Flask-based web server was created to serve inference requests.
- Two HTTP endpoints were provided:
  - GET /summary: Returns model metadata in JSON
  - POST /inference: Accepts an image and returns the predicted class

Moksh Nirvaan
COE 379L

Request Format:

curl http://localhost:5000/summary

curl -X POST http://localhost:5000/inference -F "image=@test_image.jpg"

Docker and Compose:

- The server is containerized using Docker.
- Docker image pushed to Docker Hub: moksh6/project3
- docker-compose.yml file provided for simple startup and shutdown:

```
docker-compose up -d  # start server
docker-compose down   # stop server
```

Built and tested on the class x86 VM to ensure architecture compliance.

The Alternate LeNet-5 CNN proved to be an effective architecture for binary classification of disaster satellite imagery. Through strategic adjustments (dropout, padding, normalization), the model was able to learn robust spatial features and avoid overfitting. Deployment with Flask + Docker makes it easy to use the model in real-world scenarios.

All source code, Docker files, and usage instructions are available in the GitHub repo and the included README.