Data preparation: What did you do?

To prepare the dataset for training, I first created directories for them and then copied over the images from the original dataset to the folders I created using a 80/20 split. The other 20% is for testing. Then, loaded the dataset by using Keras preprocessing image_dataset_from_directory to load images from the project folder. The folder contained two sub-directories, damaged (images of buildings with damage) and undamaged (images of buildings without damage). Then, I split the dataset into 80% training and 20% validation data to train the model and be able to evaluate it. I also printed out the number of images and how many are in the training and testing data to ensure the dataset is correct. Next, I resized the images to (150, 150) to ensure that the input dimensions were consistent. Finally, I normalized pixel values to the range [0, 1] by applying a rescaling layer and implemented data augmentation techniques like rotation, zoom, and horizontal flipping to enhance model generalization.

Finally, I printed out a few images with the label to make sure that they were what I was expecting and that there were no random pictures that were not satellite images. I also printed out sample images from each class to understand the data distribution.

Model design: Which architectures did you explore, and what decisions did you make for each?

Fully Connected ANN (Artificial Neural Network):

For the fully connected ANN, it flattens the layer to convert image data into a 1D vector. I used three dense layers, 128 neurons with ReLU activation, 64 neurons with ReLU activation, and 1 neuron with a sigmoid activation for binary classification. I think ANN is a simple approach but doesn't have the ability to extract spatial features, making it less useful for image classification. It does work well on structured/tabular data but performed worse than CNNs for this dataset. I found that using neurons that are multiples worked better and so did even neurons.

LeNet-5 Convolutional Neural Network (CNN):

The LeNet-5 CNN has two convolutional layers, one with 6 filters (5×5) followed by MaxPooling and one with 16 filters (5×5) followed by MaxPooling. It has three fully connected layers, each with 120 neurons with ReLU activation, 84 neurons with ReLU activation, and 1 neuron with a sigmoid activation for binary classification. I think that LeNet-5 is a classic CNN and performed better than the ANN due to feature extraction. I played around with the numbers and found that more is not always better and the model will overfit with more but underfit with less, thus I found some layers that stayed in the middle.

Alternate LeNet-5 Architecture:

The alternate LeNet-5 CNN followed the provided paper, which has three convolutional layers, one with 32 filters (3×3) followed by MaxPooling, one with 64 filters (3×3) followed by MaxPooling, and two with 128 filters (3×3) each followed by MaxPooling. It has dropout that has hyperparameter tuning to prevent overfitting, 512 neurons with ReLU activation, and 1 neuron with a sigmoid activation for binary classification. I think this CNN performed the best because there were more filters and smaller kernel sizes to capture more detailed patterns in images, dropout helps prevent overfitting, and deeper feature extraction. However, after running the model a few times and making small changes, I found that it was still overfitting, just not as severely as the original LeNet-5 CNN. In addition, I also found that having the hyperparameter tuning for the dropout made training the model very slow.

Model Training:

Finally, I trained each model using the training dataset and then validated it using the validation set.

Model evaluation: What model performed the best? How confident are you in your model?

After training and evaluating the three architectures, their accuracy and loss on the validation dataset are below.

Model	Training Accuracy	Validation Accuracy	Notes
Fully Connected ANN	76.98%	71.25%	Struggled because of the lack of spatial feature extraction.
LeNet-5 CNN	99.79%	92.78%	Captured spatial features but slightly overfitted.
Alternate LeNet-5 CNN	98.52%	97%	Best performance, deeper network + dropout reduced overfitting.

I think it performed better because more convolutional layers allowed for deeper feature extraction, and the 0.5 dropout reduced overfitting and improved generalization. I am very confident in this model because the model was able to achieve 97% validation accuracy, showing strong generalization to unseen data.

Model deployment and inference: A brief description of how to deploy/serve your model and how to use the model for inference (this material should also be in the README with examples)

After training the CNN, the first step for deployment is to save the trained model, which I did using TensorFlow's built-in **model.save()** method. This stores the complete model, including the architecture, weights, and optimizer configuration, to a directory.

Once the model is saved, it can be deployed locally or in the cloud. I first tested to make sure the model worked by restarting the kernel and then testing the dataset on the saved model. In order to deploy it, I created a Flask server. The API would load the saved model and expose an endpoint that accepts image input and returns the model's prediction. When an image is sent to this endpoint, the server resizes and normalizes it before passing it through the model for inference. This setup allows me to integrate the model into applications or web services.

Docker Image Name:

el32859/ml-housing-api

Docker Build:

docker build -t el32859/ml-housing-api .

Docker Run:

docker run -it --rm -p 5000:5000 el32859/ml-housing-api

Docker Compose:

docker compose up

How to use model:

- **Model File**: Stored at *models/best_damage_classification_model.keras*
- **Input**: A JPEG/PNG image of a building after a hurricane.
- **Output**: A prediction of "Damage" or "No_Damage" with a confidence score between 0 and 1

Running the Server:

To start the Flask server, simply run: *python3 api.py*

This will start the API at *http://localhost:5000*.

API Endpoints:

Provides basic model information: curl http://localhost:5000/summary

Response should look like this:
{

"version": "v4",

"model_name": "Alternate LeNet-5 CNN",

"input_shape": "(150, 150, 3)",

"output_labels": ["damage", "no_damage"]

}

Accepts an image and returns a prediction: curl -X POST http://localhost:5000/inference \

-F "image=@no_damage.jpeg"

Response should look like this:
{

"prediction": "Damage",

"confidence": 0.123456

}

References:
Alternate LeNet-5 CNN: https://arxiv.org/pdf/1807.01688.pdf
Dropout Layer:
https://stackoverflow.com/questions/47892505/dropout-rate-guidance-for-hidden-layers-in-a-convolution-neural-network
ChatGPT: For this project, I had a hard time deploying the interference server. Therefore, I used ChatGPT to help me with debugging the deployment. My main issues were that I already had something running on port 5000 and, therefore, cannot deploy again. So I had to go in and terminate what was already running so that I could deploy again. Another issue that I had was that I was not able to pipe the images into my server. Therefore, ChatGPT helped me debug my **api.py** code so that it would be able to work with raw image files.

After I deployed the model I realized that it was unable to classify things correctly. This confused me a lot because I thought I deployed and trained the model correctly. It turned out that I was training on the normal dataset and not the rescaled dataset. Therefore, when I try to use it on a rescaled image, it would not do very well.