Data Preparation:

The data preparation phase involves three steps: loading, validating, and normalizing. For the loading step, I load the pictures folder from GitHub. Under the pictures folder, there are two folders: damage and no_damage, each containing the corresponding labeled pictures. Then I use the “image_dataset_from_directory” method from TensorFlow to convert pictures to a labeled image dataset. I further split the dataset into training, validation, and testing datasets while ensuring the data inside has the same distribution. For the validation step, I checked the class label in the dataset; additionally, I displayed images from the dataset to ensure they are loaded correctly. For the normalizing step, I rescale each picture’s pixel value to [0,1] to keep training numerically stable.

Model Design:

I used three models: dense ANN, LeNet-5 CNN, and the Alternate-LeNet-5 CNN.

In the first model, I limit the picture size to 64*64 so that the ANN won’t exceed the memory limit while training. The ANN model contains four layers; the first layer is a fully-connected layer that connects to every input. The second and third layers are hidden layers that contain 4096 and 128 neurons to learn nonlinear combinations of image pixels. The final layer outputs 2 classes with softmax to produce probabilities that sum to 1. I use ‘Adam’ optimizer and validate the result with ‘accuracy’ metrics.

The LeNet-5 CNN model contains 5 layers, and treats the picture to be size of 150*150. The first layer is a convolutional layer with 6 filters of size 5*5, followed by average pooling. The second layer is a convolutional layer with 16 filters of size 5*5, followed by average pooling. Then, the third is a flattened layer, and it feeds the feature to the subsequent two fully connected layers. Finally, the output layer produces the output with 2 classes. I use ‘Adam” optimizer and validate the result with ‘accuracy’ metrics.

The Alternate-LeNet-5 CNN model contains 9 layers, and treats the picture to be a size of 150*150. The first is an input layer, and the next 4 layers are convolutional layer with 32, 64, 128, 128 filters and MaxPooling. Then the model flattens the features, drops out 50% of the features during training to prevent overfitting. Thereafter, it uses 1 fully connected layer and the output layer that outputs a single probability [0,1]. Since the output is a probability, I use both accuracy and AUC score to measure the model, and I’m using ‘adam’ with 0.0001 learning rate as the optimizer.

Model Evaluation:

The Alternate-LeNet-5 CNN performs the best with the highest testing accuracy among all three models. I'm confident in the Alternate-LeNet-5 CNN model since it also has a 0.99 testing AUC, which shows its robust ranking ability. Moreover, it explores 3.45 million parameters and also uses dropout to reduce overfitting. Therefore, I'm confident in its ability to identify damaged houses.

Model Deployment and Inference:

The model can be deployed locally by first pulling the image from Docker Hub:
docker pull jiajuwang/damage-detection:latest
Then it can be run on port 5000 by this command:
docker run -d -p 5000:5000 --name damage-detection-server jiajuwang/damage-detection:latest

These are two endpoints and example responses:

GET/summary
response:

```
{
   "status": "ok",
   "image_size": "150x150",
   "class_names": ['damage', 'no_damage'],
   "model_type": "Alternate-Lenet-5 CNN architecture"
}
```

POST/inference
header:

```
{
   files: {
      image: BufferedReader[_BufferedReaderStream]
   }
}
```

response:

```
{
   "prediction": predicted_class,
   "probabilities": {
      "damage": prob_damage,
      "no_damage": prob_no_damage
   }
}
```