

Project 03

# Analyzing Hurricane Harvey's Impact with Machine Learning and Satellite Images

---

Oswaldo Salinas

Serena Shah

10th April, 2024


## Data Preparation

In the project focused on classifying satellite images of buildings as damaged or not damaged following Hurricane Harvey, a multiple step approach was taken to prepare the dataset for machine learning model training. The initial step involved setting up structured directories for training and testing datasets using the ``os`` and ``pathlib.Path`` modules, ensuring that a clear organizational framework was established for the image data.

The `os.listdir` function is used to gather lists of filenames from two directories: one containing images of damaged structures (`data_all_modified/damage`) and the other containing images of structures without damage (`data_all_modified/no_damage`). These lists are stored in the variables `damage_all_paths` and `no_damage_all_paths`, respectively. For both categories, 80% of the image paths are randomly selected to form the training set. This is achieved using the `random.sample` function from an imported `random` module, which ensures a random selection of images without repetition. The remaining 20% of images, not selected for the training sets, are assigned to the testing sets. This is determined by a list comprehension that includes an image path in the testing set if it is not found in the corresponding training set. To confirm that the splitting process has been correctly executed, a check is performed to ensure there are no overlapping images between the training and testing sets for both categories. This is done by checking for any image paths present in both the training and testing lists and printing the count of such overlaps, which in this case, is 0 for both categories. This is important because we will test against unseen data.

The next step is to copy image files from a source directory to destination directories arranged to separate training and testing sets, as well as damaged and non-damaged categories. After importing the `shutil` module, which provides a collection of high-level operations on files and collections of files, `shutil.copyfile` is used to copy the contents of a file from one location to another. This is done by first defining two key directory paths: `root_dir` as `'data_all_modified'`, the repository containing the original dataset categorized into `'damage'` and `'no_damage'` subdirectories, and `split_root_dir` as `'data'`, designated as the foundational directory for structuring the training and testing directories. Iterating over the prearranged lists of image paths for training and testing across both damage categories, the script uses `shutil.copyfile` to transfer each image from its original position in `root_dir` to a new, specified location within `split_root_dir`. These destination paths are made to sort out whether an image is part of the training or testing set and whether it is classified under `'damage'` or `'no_damage'`. This methodical organization not only streamlines the arrangement of the dataset but also primes it for the subsequent phases of model training and evaluation, ensuring images are readily accessible based on their designated categorizations and splits.

After completing the copying operations, the next verification step ensures all images are accurately copied to their designated directories. By enumerating the contents of each target directory for both training and testing



datasets across 'damage' and 'no\_damage' categories with ``os.listdir``, and then counting and reporting the number of files, this process confirms the integrity of the data transfer. This essential verification step guarantees the dataset's integrity and readiness for the next phases of the project.

Following the organization and splitting of the data, the next phase involved preprocessing the images to make them suitable for training. The process begins with importing **TensorFlow** and the Rescaling layer from **TensorFlow Keras** to handle and preprocess the image data. Directory paths for both training (**train\_data\_dir**) and testing (**test\_data\_dir**) datasets are defined, in preparation for data manipulation. For training purposes, images are processed in batches of 32, with each image size being 128x128 pixels; a similar image size is maintained for testing, albeit with a smaller batch size of 2 to accommodate resource constraints or testing requirements.

Next is to load and categorize training data using ``tf.keras.utils.image_dataset_from_directory``, scanning the specified directory for images and inferring class labels from the subdirectory structure. 20% of this data is allocated for validation with the ``validation_split`` parameter, ensuring a clear separation between training and validation datasets. A random seed ensures reproducibility across runs. After segmentation, a Rescaling layer normalizes pixel values to [0, 1], optimizing model training efficiency. This normalization applies to both training and validation datasets, resulting in ``train_rescale_ds`` and ``val_rescale_ds``. The test dataset, loaded from ``test_data_dir``, undergoes similar rescaling for consistency, producing ``test_rescale_ds``. This comprehensive data preparation, from loading to normalization, establishes a strong foundation for neural network training and evaluation, ensuring optimal processing of images for all use cases.

## Model Design

### Artificial Neural Network (ANN)


The first model is an Artificial Neural Network (ANN) using **Keras**. An ANN is a computational model inspired by the human brain's structure and function. It consists of layers of interconnected nodes, called neurons, which process input data to perform tasks like classification, regression, or pattern recognition. ANNs learn from data by adjusting the connections (weights) between nodes through training, enabling them to model complex, nonlinear relationships within the data.

The Artificial Neural Network (ANN) for this project uses a sequential architecture. It kicks off with a Flatten layer, transforming input images (each with height, width, and 3 color channels) into a 1 dimensional array to serve as the foundation for processing. The network then progresses through two **`Dense`** layers, the first hosting 120 neurons and the second 128, both employing the **`relu`** activation function. These layers are tasked with extracting features and performing nonlinear transformations on the input data. The final layer is a **`Dense`** layer with a single neuron, using the **`sigmoid`** activation function, and it outputs a probability indicating the likelihood of the input belonging to one of the classes. The model is compiled with the **`adam`** optimizer and **`categorical\_crossentropy`** loss function.

The model is trained using the **`fit`** method, which takes in the training data **`train\_rescale\_ds`**, batch size **`32`**, number of epochs **`20`**, and the validation data **`val\_rescale\_ds`**. The **train\_rescale\_ds** dataset is used to train the model over 20 epochs. During each epoch, the model learns by adjusting its weights to minimize the loss function, using the specified batch size for each iteration of the training process. The **validation\_data** parameter is set to **val\_rescale\_ds**, which allows the model to evaluate its performance on a separate dataset not used during the training process. This helps in monitoring for overfitting and assessing the generalization capability of the model.

### Lenet-5 Convolutional Neural Network (CNN)

LeNet-5 is a convolutional neural network (CNN) model designed for classifying images into three categories. It features a sequential architecture that processes images through a series of convolutional and pooling layers before making classifications through fully connected layers. This project uses Keras to make convolutional and pooling layers for feature extraction, followed by fully connected layers for classification. The model begins with a convolutional layer that has 6 filters, each of size 3x3, using **'relu'** as the activation function. This layer is designed to extract basic features from input images of size 128x128x3 (height, width, color channels). Following the convolutional layer is an average pooling layer with a pool size of 2x2, which reduces the spatial dimensions of the feature maps. The second convolutional layer consists of 16 filters of size 3x3, again followed by **'relu'** activation and an average pooling layer. This layer aims to extract more complex features from the output of the previous pooling layer. After extracting features through convolution and pooling, the feature maps are flattened



into a 1 dimensional vector to serve as input for the subsequent fully connected layers. Flattening is necessary to transition from the 2 dimensional feature maps to a format suitable for dense layers. The first fully connected layer is densely connected with 120 neurons and uses **'relu'** activation. It's responsible for further processing the features extracted by the convolutional layers. Another densely connected layer follows, this time with 84 neurons and **'relu'** activation, which continues the pattern of feature processing. The final layer is a dense layer with 3 neurons (because there are three target classes) and uses the **'softmax'** activation function. The **'softmax'** function outputs a probability distribution over the classes, making it suitable for multi-class classification tasks.

In the training phase of the LeNet-5 model, the process is done using the `model_lenet5.fit` function, using `train_rescale_ds` as the primary training input. The model training is structured to process mini-batches of 32 images per iteration to optimize memory utilization but also in an attempt to accelerate the convergence rate of the model's learning algorithm. The training regimen spans across 20 epochs. At the same time, the model's generalization capability is evaluated against a separate validation dataset, `val_rescale_ds`, to find the model's performance on data not previously encountered during the training process. This is done to identify and mitigate overfitting. Upon completion, the `history` object encapsulates a detailed record of the training and validation accuracy and losses across all epochs to show the model's learning trajectory.

### Alternative Lenet-5 Convolutional Neural Network (CNN)

The Alternate-Lenet-5 model used in the project is similar to the Lenet-5 architecture but it is modified. The Alternate-Lenet-5 architecture is more complex, featuring more and progressively increasing filters across its convolutional layers (starting from 32 up to 128 filters) compared to Lenet-5, which starts with 6 filters and increases to only 16. This allows Alternate-Lenet-5 to extract a broader and more complex range of features, which is crucial for identifying subtle differences between damaged and undamaged buildings. Additionally, Alternate-Lenet-5 employs max pooling, known for preserving dominant features, while Lenet-5 uses average pooling, which may dilute important feature details. Another significant modification in the Alternate-Lenet-5 is the inclusion of a dropout layer to combat overfitting by randomly dropping neurons during training, a feature absent in Lenet-5. This could make Alternate-Lenet-5 more robust and generalize better on unseen data.

Ultimately, the Alternate-Lenet-5's complex and feature-rich design may offer superior performance for the nuanced task of damage assessment in satellite imagery, at the cost of increased computational demands. In contrast, Lenet-5's simpler, more efficient architecture might be less capable of capturing the detailed features necessary for high accuracy but offers a less resource-intensive option. The choice between them hinges on the balance between computational resource availability and the need for model complexity to achieve the project's objectives.

## Model Evaluation

Evaluating three models for classifying satellite images into damaged or undamaged buildings reveals significant insights into model selection for image-based machine learning. The ANN model, characterized by its simplistic design consisting primarily of dense layers, showcases the highest number of trainable parameters at approximately 5.9 million. This extensive parameter space, while indicative of a high learning capacity, also poses a significant risk of overfitting and necessitates considerable computational resources. When looking at each epoch throughout the training process, the ANN model demonstrates limited efficacy in accurately classifying the images, as evidenced by its plateauing validation accuracy early in the training sequence and a final test accuracy of only 66.39%. This performance shortfall highlights the model's inherent limitations in capturing the spatial and hierarchical complexities inherent to satellite imagery.

Conversely, the Lenet-5 model introduces convolutional layers into the architecture, using average pooling and a modest filter count in its convolutional layers. With around 1.74 million parameters, it strikes a balance between model complexity and computational efficiency. Throughout its training, Lenet-5 exhibits a consistent improvement in learning, as reflected by a steadily increasing validation accuracy after each epoch and culminating in a substantial final test accuracy of 95.17%. This marked improvement over the ANN model underscores the convolutional neural network's (CNN) advantage in processing image data.

LeNet-5 and traditional ANNs differ primarily in architecture and application scope. LeNet-5, as a CNN, is tailored for image recognition tasks, featuring layers specifically designed for automatic feature extraction and spatial data processing, such as convolutional and pooling layers. In contrast, traditional ANN architecture consists of fully connected layers and are more versatile, handling a wide range of data types but lacking the inherent spatial processing advantage of CNNs. Consequently, LeNet-5 excels in image-related tasks by leveraging spatial hierarchies, while ANNs potentially offer broader applicability.

The Alternate-Lenet-5 model, with its deeper convolutional architecture and max pooling strategy, represents a more advanced approach. Despite having fewer parameters than Lenet-5, approximately 804k, this model demonstrates superior learning efficiency. It achieves the highest final test accuracy of 98.44%, reflecting its enhanced capability to discern complex features within the satellite images. The architecture of Alternate-Lenet-5, particularly its deeper layers and sophisticated feature extraction mechanisms, allows for a nuanced interpretation of the data, thereby facilitating highly accurate classifications.

All-in-all, the underperformance of the ANN model relative to its convolutional counterparts highlights the inadequacy of dense-only architectures in handling spatial data complexities. Meanwhile, the superior performance of the Alternate-Lenet-5 model not only showcases the benefits of a convolutional approach but also emphasizes the value of architectural depth and efficient feature extraction in achieving high accuracy in image-based classification tasks.

## Model Deployment

The deployment process involved containerizing the pre-trained model within a Docker image to ensure consistency and ease of deployment across different environments. The command `docker pull serenashah/ml-proj03-api` is used to download the Docker image named `serenashah/ml-proj03-api` from Docker Hub. This image contains the best performing model, Alternate LeNet-5, along with an inference server that can process incoming image data to make predictions. The command `docker run -it --rm -p 5000:5000 serenashah/ml-proj03-api` starts a container from the pulled image. This container runs an inference server that listens for HTTP requests on port 5000, ready to accept images for classification. The command `curl localhost:5000/models` sends an HTTP GET request to the inference server, specifically to the `/models` endpoint. This request is designed to retrieve information about the deployed model. To classify an image, the image data needs to be serialized into a format that can be transmitted over HTTP. This typically involves converting the image into a list or array of pixel values. The `input_image` variable should contain this serialized image data. The Python code snippet provided demonstrates how to send this serialized image to the inference server using a POST request: `rsp = requests.post("http://172.17.0.1:5000/models", json={"image": input_image})`. The server receives the image data, processes it through the Alternate LeNet model, and returns a prediction indicating whether the image shows a damaged or undamaged building.