Project 3: Hurricane Damage Assessment

By: Antonio Jimenez and Anna Victoria Lavelle

Data Preparation:
Our first objective was to load in the data into the correct directories and create the file paths. We randomly assigned 80% of our data into a training folder and split it up between damage and no damage folders. We also discovered that our images were 128 pixels x 128 pixels through analyzing the file properties. Then, using the tensorflow.keras functions, we split our training dataset into train and validation sets and created our test dataset from the images. After that, we rescaled our images using Rescaling(). Then, we were ready to build our networks.

Model Design:
We explored an ANN, a Lenet-5 CNN, and an Alternate-Lenet-5 CNN.

For the fully connected ANN, we built four different models that experimented with the number of layers and perceptrons. Our first model utilized just three layers while our last model utilized six. We also experimented with the output shape of each layer and the order of these layers. The first step before any of our other layers was using a flatten layer for our input. For our first model, we used only three layers with the number of perceptrons we used in our class examples as a base model. For our second model, we doubled the amount of layers and gave the layers nearest the input layer the largest number of perceptrons, other than the original input perceptrons, and gradually decreased the number of perceptrons as we got closer to our output layer which ultimately had two target labels. For our third model, we reversed this order and started with a smaller number of perceptrons after our input layer and gradually increased the number of them throughout the layers. However, this led to a big jump from our second to last layer and our final, output layer with two target labels. Our fourth model had layers with equal perceptron numbers other than our input and output layers.. Surprisingly, most models ended up being fairly comparable, even the first model with only three layers. Even more surprising, the model with only three layers had the highest accuracy and the lowest loss. Everything else had accuracies and losses that were about the same. Generally speaking, the ANN models were not our most successful.

For our Lenet-5 CNN, we built one model of ten layers. We used padding and MaxPooling throughout the layers to adjust the output sizes of each layer. Before adding the fully connected dense layers, we flattened the output of the convolutional layers, similar to the flattening we performed when building our ANN models. Our dense layers in this CNN had fewer perceptrons than in our ANN because the CNN had the additional convolutional layers, so we didn't think a higher number of perceptrons for the dense layers was necessary. However, the size of the output layer stayed the same as the ANN because we still had only two outputs. This model was pretty successful. It boasted 96% accuracy with only 10% loss, blowing our ANN models out of the water.

For our Alternate-Lenet-5 CNN, we built a model of twelve layers. Similar to our Lenet-5 CNN, we used padding and MaxPooling to control the output sizes of each layer. We flattened the output of the convolutional layers once again before starting the dense connections. For it to be our Alternate-Lenet-5 CNN, we utilized the information we gathered from the research paper. We utilized Dropout() to drop 50% of neurons to help prevent overfitting. With too many neurons, too much training data is remembered which can lead to overfitting, but randomly dropping 50% of the neurons and connections can help mitigate this issue. To compile the model we used the Adam optimizer as described in the paper as leading to 1% higher validation accuracy. Using dropout and the Adam optimizer, we built our most

accurate neural network, similar to the results seen in the paper. This model reported 97% accuracy and only 6% loss on the test data, making it our most successful model.

Model Evaluation:
After testing our six models, our Alternate-Lenet-5 CNN performed the best unsurprisingly. The losses and accuracies on the test data of each model are reported below.

Model	Accuracy	Loss	Trainable Param	Notes
ANN 1	76.4%	51.9%	6,291,842	3 layers, max 128 perceptrons other than input
ANN 2	66.4%	63.8%	103,059,074	6 layers, max 2048 perceptrons other than input, larger to smaller
ANN 3	66.4%	63.9%	8,691,074	6 layers, max 2048 perceptrons other than input, smaller to larger
ANN 4	66.4%	63.8%	6,341,378	6 layers, max 128 perceptrons other than input
Lenet-5 CNN	96.4%	10.1%	857,458	10 layers
Alt-Lenet-5 CNN	97.9%	6.2%	3,453,121	12 layers

We are very confident in our Alternate-Lenet-5 CNN model and even our Lenet-5 CNN model, but not as confident in our ANN models. When testing different ANN model structures, we anticipated the model with the most layers and going from larger to smaller number of perceptrons to be the most successful. It made sense that more layers would allow the model to most accurately funnel the outputs into our final two target levels. However, the paper's discussion on dropout reminded us why this might cause significant overfitting, and it ultimately led to a lower accuracy and higher loss than our ANN model with the fewest number of layers. Even when we drastically reduced the number of perceptrons in our ANN 4 model, it gave us comparable accuracy and loss to our overfitted models, probably because there were still too many layers. Dropout would have been an interesting tool for the ANNs and definitely contributed to the success of our Alternate-Lenet-5 CNN.

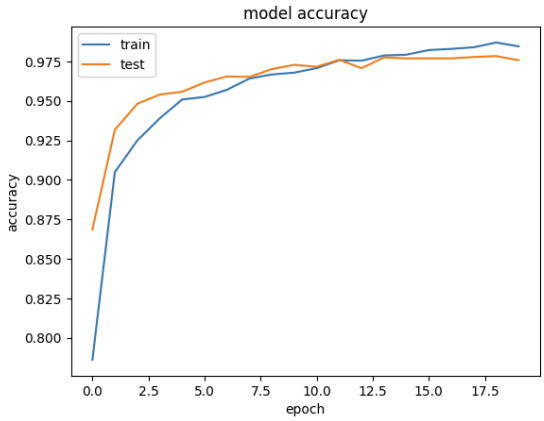

Figure: Model Accuracy of Alternate-Lenet-5

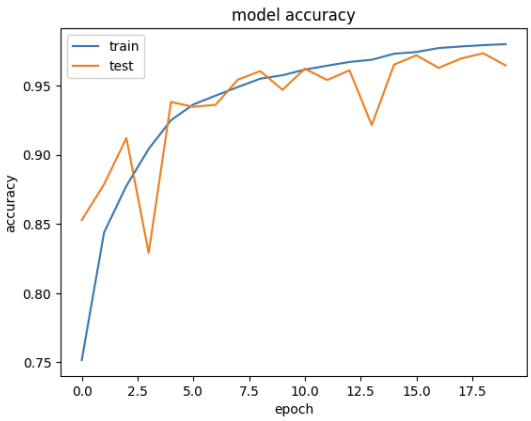

Figure: Model Accuracy of Lenet-5

When comparing our Lenet-5 and our Alternate-Lenet-5, their results of accuracy and loss were very similar, but the above plots of their accuracies differ greatly. The Alternate-Lenet-5 CNN had a steadier accuracy rating over the course of the epochs than the Lenet-5 did. As predicted by the research paper, the use of dropout and the Adam optimizer led to a higher accuracy and lower loss for the Alternate-Lenet-5 CNN. Depending on the type of project and data, the Alternate-Lenet-5 adjustments might be worth it when constructing a neural network. They certainly paid off in our case, especially in the loss of our test data.

Model Deployment and Inference:
The model utilizes the class container, as well as an additional docker image to launch the Flask application. In following the instructions to pull and run the hurricane_api docker image, the Flask application will be successfully launched. In another terminal, clone the repository, make the required directories according to the README instructions, and pull, run, and exec into the class container. Then, from the class container, follow the instructions to install tensorflow_datasets and run the test_api.py. For image -93.53950999999999_30.982944.jpeg, the result yields 99.9% chance of damage, which is true for the selected image. Users can follow the commands in the README to pass in one image at a time using the image's file name.

Users can also access information, such as the number of parameters and name, of the Alternate-Lenet-5 and Lenet-5 CNN models by using the Flask application's GET methods following the command pattern "curl -X GET 172.17.0.1:5000/models/…" from within the class container.

Additional Notes:
For this project, we used ChatGPT to learn more about tf.keras.utils.image_dataset_from_directory and how to use the map object to test only one image from our dataset to check our results. We also used it to examine the best ways to convert a JPEG file to JSON. It also helped us debug our code when we got dimension errors when converting the JSON data into a NumPy array or trying to pass our test dataset as an argument to different functions that would allow us to evaluate our data.

Our models were split into two notebook files, one for ANN and one for CNN. We found this to be most efficient for training our models because some of the models took about an hour to fit.