COE379L: Project 3 Report

Data Preparation

First, the dataset was split into training, validation, and test subsets. The training data was split 80-20 into training and validation sets using image_dataset_from_directory. All images were resized to 128x128 pixels, and the batch size was set to 32. This follows what we did in class.

Images were normalized using rescaling to bring pixel values to a [0, 1] range. This serves to preprocesses and speed up training. This rescaling was applied to the training, validation, and test datasets. The datasets were then input to the three models.

A few sample images were displayed to understand the kind of images we were working with as well as to ensure that they were being loaded correctly. I also observed the image shape, label shape, and data type.

Model Performance

Three models were evaluated: a dense artificial neural network, a Lenet-5 convolutional neural network, and the alternate Lenet-5 convolutional architecture from the following paper: https://arxiv.org/pdf/1807.01688.pdf.

ANN Model:
The ANN model served as a sort of baseline to see how well a simple dense network could perform on the task. It flattened the 150x150 RGB image into a 1-dimensional vector and then passed that through two dense layers. First there was a dense layer with 784 units with ReLU activation. Next, there was a dense layer with 128 units with ReLU. Finally, there was an output layer with 2 units with softmax that was used for classification. The model was compiled with the Adam optimizer and sparse categorical cross-entropy was used as the loss function. It was trained for 5 epochs.

Lenet CNN:
The Lenet CNN model was built with multiple convolution and pooling layers. I had one convolution layer with 6 filters that used average pooling, another convolution layer with 16 filters that used average pooling, then the data was flattened and passed through two

dense layers, and finally output in a layer with 2 units and softmax. The model was compiled with RMSprop with a learning rate of 1e^-4 and trained for 20 epochs. This architecture is known to be simple and effective with small mage tasks. It was able to better capture spatial features in the images and outperform the ANN.

Alternate Lenet CNN:
The alternate LeNet model was built using a similar CNN architecture but with some adjustments to be a deeper CNN. It had a convolutional layer that had 32 filters, rather than 6, and used max pooling rather than average pooling. Then there was another convolution layer with 64 filters, also with max pooling. This was to help the model detect more complex patterns in the data. After flattening the output, there was a dense layer with 128 units and ReLU activation, that then finished with a 1-unit output layer with sigmoid. This version was also compiled with RMSprop and a learning rate of 1e^-4, and trained for 20 epochs at first. Changes had to be made regarding the input shape last minute during class, and training 20 epochs generally takes 2+ hours on my machine, so when I trained the edited model, it was only trained for 2 epochs. For the purpose of this report, I will be referencing the version trained with 20 epochs as the only difference was the input size. Compared to the original LeNet, this architecture was more able to analyze higher-level features and had a higher accuracy, likely because we used more filters and max pooling instead of average pooling.

Model Evaluation

The ANN model had the lowest validation accuracy at around 70%. It was likely less able to capture spatial patterns due to the lack of convolutional layers. It was less able to generalize and had limitations with visual features. The Lenet CNN performed significantly better, reaching around 90% validation accuracy. The convolution and pooling helped to recognize spatial patterns in the images, and the results were consistent across training and validation sets, meaning it generalized better than the ANN. The best performance was from the alternative Lenet CNN, which reached about 94% validation accuracy. It had smoother loss and accuracy curves. The architecture was deeper and used more filters, allowing it to learn richer representations without overfitting. Based on the results, I'm most confident in the alternative Lenet CNN's predictions since it performed the best overall and remained stable across all 20 training epochs.

Model Deployment and Inference

The API has two endpoints: /summary for information about the API and /inference for taking in an image file and returning the prediction ("damage" or "no damage").

For deploying using Docker, first clone the repository. Next, build and start the container using **“docker-compose up –build.”** The API should be available at http://localhost:5000. To stop the container, run “**docker-compose down.”**

Example requests:

1. /summary: uses GET and returns the API version, name, description, and number of parameters for the model
    - curl request: curl http://localhost:5000/summary
    - json response:
      ```
      {
        "version": "v1",
        "name": "damage",
        "description": " ----- Categorize damage from hurricanes ------",
        "number_of_parameters": 1234567
      }
      ```
2. /inference: uses POST to accept an image file as input and return the prediction (“damage” or “no damage”)
    - curl request: curl -X POST http://localhost:5000/inference \
      -F "image=@path_to_your_image.jpg"
    - json response with success:
      ```
      {
         “prediction”: “damage”
      }
      ```
    - json response with error:
      ```
      {
        "error": "Invalid request; pass a binary image file as a multi-part form
      under the image key."
      }
      ```

Resources

Notes from the COE379L website were used for informing the models and ChatGPT was used for help with deploying using Docker and for informing my knowledge on what different parts of certain layers meant.