Saood Usmani's Report
Project 3

Data preparation: what did you do? (1 pt)

- To prepare the data, I had to first remove the train and test directories in case they existed before the notebook was run for consistency. Then, I created these directories, and within them created subdirectories for our two labels, damage and no_damage. After that, I got all the paths from our damage and no_damage directories, storing them in two arrays, and verified their count by printing out the length of the respective array. I also visualized these counts using matplotlib, seaborn, and pandas. I found that there were almost twice as many damage images compared to no damage images, which might introduce biases into my model.
- Now that I had the paths successfully in my notebook, it was time to split them into the subsequent train and test directories that I had created earlier. I decided to use an 80% 20% split, where 80% of my data went into the train directores, and 20% of my data went into the test directories. I did this by using the random sample method. First I randomly picked 80% of my data and put it into the train path array, and then any data that was not picked, the leftover 20%, I put into the test path array. I also checked for overlap to make sure there wasn't any path that accidentally got into both the train and the test directories, which there wasn't. Once I was done making the arrays, I simply copied these paths into their respective directories i.e. train damage paths went into the data/train/damage directory. One particular thing I had to check for was to make sure that the path wasn't an .ipynb_checkpoints directory, as if it was it would try to treat that directory as a file and copy it over to our train and test directories, which would not only corrupt our data set but also cause the program to crash.
- Once I had set up the train and test directories, it was time to make the keras train dataset, validation dataset, and test dataset. I did this by using the keras.utils.image_dataset_from_directory method. I also applied some rescaling to the data to make all values in between 0 and 1 by using the Rescaling class.
- Now, I was almost ready to run my models. However, one thing I had to do first was convert these datasets to x and y numpy arrays for my ANN. I did this by creating a custom dataset_to_numpy method, which took in a dataset and returned two numpy arrays, one for the x array and one for the y array. This method went through all the batches in the dataset, adding their images and labels to my images and labels array as one list. Then once all the batches were added, I combined all the individual lists for both the images and labels into two big lists and returned them. Using this method, I converted my train, validation, and test datasets into X_train, y_train, X_val, Y_val, and X_test, Y_test arrays. Finally, I flattened these arrays for my ANN since ANNs can only accept one dimensional arrays. Now, I was finally ready to run my models.

Model design: which architectures did you explore and what decisions did you make for each? (2 pts)

- I explored the dense ANN architecture, the LeNet-5 CNN architecture, the Alternate-Lenet-5 CNN architecture, and the VGG-16 CNN architecture.
- For the ANN architecture one key decision I had to make was setting the output layer to one perceptron, since there we were performing binary classification. Because of this, I

also had to use the sigmoid activation function, since softmax is for multi label classification. I also tried adding more layers, as I added another 784 perceptron layer in the middle of my network. However, I found that this may have hurt my accuracy as I had a terrible accuracy of .33, but that could be because I had forgotten to switch my activation function from softmax to sigmoid. Regardless, I removed this layer and switched the activation function, resulting in a decent accuracy of .72. I would have ran this test again time permitting with the correct activation function, but honestly the improved accuracy wasn't that great and I didn't want to waste my time on improving a sub 80% accuracy when I had models achieving 90%+.

- For the LeNet-5 CNN one key decision I had to make was setting the output layer to two perceptrons. This confused me a bit at first, as I thought we would only need one perceptron since we were performing binary classification. However, it appears to me that when training with datasets, you need the amount of perceptrons equal to the number of classes/labels that you have, instead of what we're familiar with of the number of classes/labels -1. I'm not completely sure why this is the case, but I am quite confident this is correct as when I tried to decrease the amount of perceptrons from two to one I got an error during fitting. Other than that, my LeNet-5 CNN was the same as the one shown in class.
- For the Alternate-Lenet-5 CNN, I also had to make the key decision I made above for the LeNet-5 CNN architecture. This made my CNN diverge from the paper a bit, as my very last output layer had twice as many trainable parameters (513 vs 1026). Other than that, my Alternate-Lenet-5 CNN was the same as the one presented in the paper.
- For the VGG-16 CNN, I again also had to make the same key decision as I made in my LeNet-5 CNN. However, I also made a lot of key improvements to attempt to boost the test accuracy of my model to 98%. Some of these improvements involved:
    - Adding a data augmentation layer that applied random flips, rotations, and zooms directly to the model, without the use of a pipeline.
    - Unfreezing the last few layers of the VGG16 model to enable fine-tuning.
    - Improving my dense layers by adding batch normalization and dropout layers to help learn complex patterns while reducing overfitting.
- Finally, the last key decision I made for every model was running 40 epochs. Although very time consuming, I believe this did greatly improve my models' accuracy since I believe it boosted them by a critical 1% or 2%.

Model evaluation: what model performed the best? How confident are you in your model? (1 pt)

- The model that performed the best was the Alternate-Lenet-5 model. I was pretty surprised by this, as I thought the model that would perform by far the best would be the VGG16 model as it is the most complex model. However, the Alternate-Lenet-5 CNN had an accuracy of .97 while the VGG-16 CNN was quite close to it with an accuracy of .968. I am very confident in my model, as such a high accuracy is very promising and honestly something I have not gotten before. Although I believe this makes sense, as it shows the power of neural networks whereas before we had been using simpler models. I ended up using VGG-16 for my inference server as I believe the Alternate-Lenet-5 model might be slightly overfit, and I think that VGG-16 will perform better for unknown 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) (1 pt)

- To deploy/serve my model, simply pull my docker image from docker hub with docker pull usmanis0115754/ml-structures-api:latest. Also, download the docker-compose.yml file in the same directory from my github repo. Then, run docker compose up -d to start the container. Now, once the container is started, you can easily run the ./start_grader.py file or can use curl to test the server. Here are some examples of using curl for both the get and post commands (note, you will probably have to change the image path for the curl command)
    - curl -X POST http://127.0.0.1:5000/inference -F "image=@data/damage/-93.66109_30.212114.jpeg"
        - {
        - "prediction": "damage"
        - }
    - curl localhost:5000/summary
        - {
        - "description": "Classify images based on if they contain damage or no damage",
        - "max_number_of_parameters": 14714688,
        - "name": "damage or no damage",
        - "version": "v1"
        - }
- To stop the container, simply use the command docker compose down, and the container will stop.

ChatGPT usage:
I used ChatGPT more than I did for the last few projects since there were many new concepts/resources that I was trying to explore. I used it for:

- Helping swap out variable names
- Debugging the .ipynb_checkpoints error
- Converting our dataset data into numpy arrays for the ANN
- Docker help
- Flask API help
- Data Visualization
- VGG16 improvements
- General sanity checking

If you want to see exactly how I used it, here is the GPT thread https://chatgpt.com/share/67f7eed4-4c28-8013-8036-2081e32d7b73. Whatever new concepts/technologies it showed me I always tried to understand exactly what they were doing and make sure I was learning these concepts.