Hurricane Damage Classification: Neural Networks

Jakob Long, Ethan Colley
COE379L

Introduction:

In this project we were tasked with using what we learned about deep learning and neural networks to take a dataset of images, refine it, then use the data to create neural networks that can accurately classify damaged buildings and undamaged buildings in the aftermath of a hurricane.

The dataset itself was a dataset full of images taken after hurricane Harvey with separate folders for damaged and undamaged buildings. If able to correctly classify this data, machine learning could be used in a multitude of ways to help accurately detect things like people in need after a natural disaster. The goal of this project is to provide 3 neural networks that can classify the images and then find which neural network is the most accurate and persist the most accurate model to disk so that it can be easily used in the future.

The three neural networks used in this project were the ANN model or the artificial neural network, the LeNet-5 architecture, and a modified LeNet-5 architecture from a past research paper we were given. Each model will be evaluated based on overall accuracy and validation accuracy.

Data Preparation:

This dataset was a set with 17057 image files split into two folders: one for damaged buildings and one for non damaged buildings. No conversions or anything needed to occur since all data was the same type. The only preliminary steps taken before preprocessing were to make sure that the image directories were clean and to ensure that they existed. Once these steps were taken, we could move on to the preprocessing steps.

Code for Python Structures:

- To properly ‘structure’ our data, we implemented code that’s seen within the Dataset construction section of Jupyter notebook. These scripts ensure that the folders that the testing & training data will reside in exist & have an available path. We then split the data from the original ‘damage’ & ‘no damage’ folders, randomly using the random library. Overlap is then checked for all directories. Lastly, we counted each of the files that resided in each directory.

Determine Basic Attributes of Images:

- The basic attributes of the images that was checked were the dimensions. For this we used functions to run over ALL of the images that resided in the train dataset folders.

Ensure Data is Split:

- Ensuring the data was split occurred in the dataset construction section, and additionally we counted the number of files that were to be used for testing, training & validation in the Data Pre-Processing section.

Image Manipulation:

- We rescaled image pixel count from [0, 255], to [0,1] to ensure that all values are within the usable range for neural networks.

Model Design, Training, Evaluation:

For this project we were given three models to deploy and try, and find which model was the most accurate. These models were the ANN model, the LeNet-5, and the modified LeNet-5 given in the research paper.

Decisions for each:

1. **ANN:** For our ANN model, we led with flattening the data immediately due the large number of parameters due to the shape being 128x128x3. This was then followed by six dense hidden layers starting from 256 perceptrons down to 8, decreasing by half in each following layer. This was to reduce the total number of parameters as we approached the total output layer, which is a binary classification, thus using only two perceptrons.
2. **LeNet-5:** The LeNet-5 model had no changes other than the input shape. As our image was in color and they were 128x128 pixels. In addition to changing the input shape, our output was changed to a binary classification so our final output layer only needed two neurons.
3. **Modified LeNet-5:** The base LeNet-5 was obviously based upon the original LeNet-5 with modifications made based on the report provided to us. We elected to use the same input layer as that of the original LeNet-5 and after this initial layer all other layers were directly following those given in the research paper. The original input layer had 6 filters, and the subsequent convolutional layers had 32, 64, 128, and 128 filters respectively. Each convolutional layer was followed by max pooling, and then after the final convolutional layer there were flattening and dropout layers, and finally two fully connected layers.

Model Evaluation:

For the three models, we plotted the performances of each epoch's testing accuracy against the validation accuracy

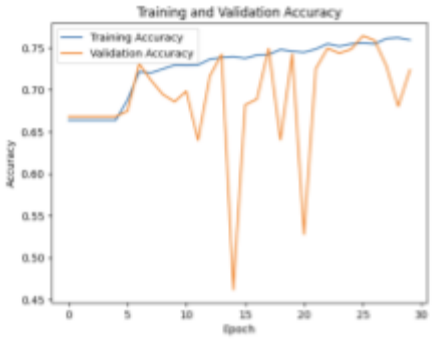


Fig 1 - Artificial Neural Network (ANN)

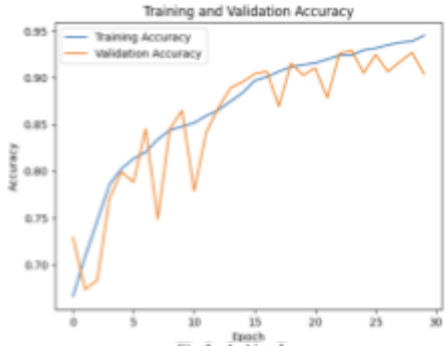
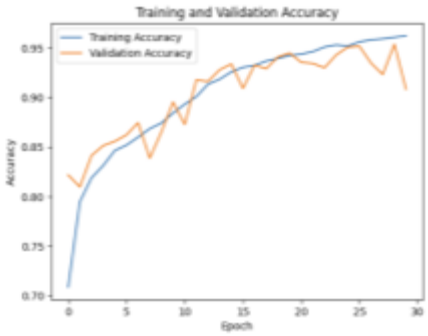


Fig 2 - LeNet-5

Fig 3 - Modified LeNet-5



From the above we're able to see that our modified LeNet-5 model had the strongest performance and had the least risk of being an overfit model according to the validation test set that was used. Additionally, when the model was used on our separate test data set we got an accuracy performance of 0.912, with a test loss of 0.21, compared to the 0.9623 accuracy the model had on the testing dataset. With only a 5% discrepancy between the two accuracies, and with the plot demonstrating that the model is not drastically overfit, we have strong confidence in our models performance.

Model Deployment:

To deploy the model and run the inference server, we must first run commands to actually access the server in docker. First the docker command "docker pull ecolley3/ml-damage-api" to pull the docker image. To run the image we must then use "docker run -it --rm -p 5000:5000 ecolley3/ml-damage-api". Once the model has been run you can then use it for inference. To do this, we must first open a new shell and run the command "curl localhost:5000/proj_models/ann/v1", which will display the information associated with the model. To see the prediction open the inference file and run the code block "rsp = requests.post("http://172.17.0.1:5000/proj_models/Mod_LeNet5/v1", json={"image":l})" and "rsp.json()" to see results.