Software Design Project 3 Report

In this project, I was tasked with building 3 neural networks on different architectures. The goal with these NN was to classify satellite Images from Texas after Hurricane Harvey as damaged or undamaged. As with every machine learning model, we want the one with the best accuracy when. The more accurately we can determine whether houses are damaged or not with intelligent systems the more efficiently we can get them help and manage damage control.

Data Preprocessing

As mentioned earlier, I am given a dataset full of satellite images from Hurricane Harvey split into damaged and not damaged folders. To process my data for my different neural networks I need to do some things differently depending on the model. However for every model I must first split the data into training/testing folders to which I used an 80/20 split!

```
Files in Train/No_Damage:  5721
Files in Test/No_Damage:  1431
Files in Train/Damaged:  11336
Files in Test/Damaged:  2834
```

Figure 1: Output representing the number of images in each folder.

Specifically for the Artificial Neural Networks (ANN) model, you need to flatten the images, to avoid any loss of information you want to resize/flatten the images with the proper dimensions. Before my image preprocessing, I dug a little into the physical properties of a few of the images in both folders to which I found their dimensions to be 128*128 pixels. Seeing as they were colored images, they fall under the RBG value dimension so all in all I classified them as (128,128,3). The goal was to make the data into a 1D array.

Whereas the Convolutional Neural Networks (CNN) models require some normalizing with the Rescaling function from one of the Keras packages. The rescaling function does something similar to what I did for the ANN model, but simply I scaled the images down to [0,1] to allow for the deep-learning models to read the images better. Unlike the ANN model CNN models are able to intake 2D arrays!

Model Architectures

For this project I used 3 different model architectures: a dense ANN, Lenet-5 CNN, and an Alternate-Lenet-5 CNN described in this paper on the 12th page. For the most part I stuck to what we had practiced during lectures when building the models.

For my ANN model I used the Sequential model with 3 dense layers. My first layer had 784 perceptrons using the 'relu' activation function, my 2nd was the same 'relu' activation but had 128 perceptrons! For the last layer I added a 2 perceptrons with 'softmax' activation although it doesn't seem very necessary seeing as softmax is more effective for datasets with multiple classes whereas I only had 2.

For the Lenet-5 Model I used a bit more layers, which is expected for CNNs. I added 9 layers: 3 'Conv2D', 2 'AveragePooling2D', 3 'Dense', and finally one 'Flatten'. My Convolution layers had perceptrons of 6, 64, and 16 respectively with pooling layers of size (2,2). My 3 dense layers had perceptrons of 120, 84, and 3 with activations of 'relu'.

Table 1 Convolutional neural network architecture that achieves the best result.

Layer type	Output shape	Number of trainable parameters
Input	3@(150x150)	0
2-D Convolutional 32@(3x3)	32@(148x148)	896
2-D Max pooling (2x2)	32@(74x74)	0
2-D Convolutional 64@(3x3)	64@(72x72)	18,496
2-D Max pooling (2x2)	64@(36x36)	0
2-D Convolutional 128@(3x3)	128@(34x34)	73,856
2-D Max pooling (2x2)	128@(17x17)	0
2-D Convolutional 128@(3x3)	128@(15x15)	147,584
2-D Max pooling (2x2)	128@(7x7)	0
Flattening	1x6272	0
Dropout	1x6272	0
Fully connected layer	1x512	3,211,776
Fully connected layer	1x1	513

Figure 2: Table my Alt-Lenet-5 model is based off of.

As seen above I was asked to base my alternative-Lenet-5 model on the table above. It is somewhat similar to my Lenet-5 model with an addition of one Convolutional layer, 4 max pooling layers instead of any average, as well as a dropout layer.

Model Evaluation

To everyone's surprise (no one) the ANN model performed very poorly in comparison to both the Lenet models. To my personal surprise my model

performed better than I expected once having gotten a result for the alternative model.

```
test_accuracy: 0.6644783020019531
test_accuracy: 0.9470105767250061
test_accuracy: 0.96858149766922
```

Figure 3: Outputs representing the accuracy results of the ANN, Lenet-5, and Alt-Lenet-5 respectively.

As seen above my Lenet-5 model had a ~95% accuracy while the alternative had ~97% accuracy which if it comes down to a choice the obvious one is the alternative seeing as the higher the accuracy the better!! Although I am rather confident in the model based off of the lecture notes, although the ANN model is rather disappointing despite the lack of high expectations. It really does just prove that when limiting/changing data to a smaller scale for proper deep learning processing it causes a lot of loss.

Model Deployment

For my deployment and inference I made a 4 new files, a Dockerfile, api.py file, and a docker-compose file and a keras model file. My keras model file is alt-lenet-5 model being used to classify the images. The api.py file has routes that explains what the model does, and allows for an input from the user to classify an image they give to the model. My Dockerfile is to build the container to allow for the use of my api inferences whereas the docker-compose is for the same but allows ease with running the container in the background!

Conclusion

This project was very insightful seeing as I didn't expect to learn how to use ML with image processing and predictions. The CNN results were very exciting to see as the end goal for this dataset was to classify REAL satellite images from after Hurricane Harvey. As I mentioned at the beginning with the use of ML for things such as these it helps make the lives of first-responders at least a bit easier and efficient with the chaos they face.