Constantinos Skevofilax
Nikhil Sharma

COE 379L Project 3 Report

Project Problem Introduction
In this project, we were given a dataset of satellite images of houses that were impacted by Hurricane Harvey. We were tasked with classifying the dataset into damaged or not damaged sets by using multiple neural networks to provide our analysis on the integrity of the houses.

Data Preparation
To prepare the data for model training, we downloaded the hurricane dataset from the class Github repository and randomly split the image paths of the damaged and undamaged images in a 80% train, 20% test split. Afterwards, we used the TensorFlow Keras model to train the training and testing data, with batch sizes of size 32 and 2 respectively.

Model Design
We explored the Artificial Neural Network (ANN), LeNet-5 CNN Architecture, and Alternate LeNet-5 CNN Architectures for this project.

With the ANN model, we decided to use 6 layers, with the output layer being a sigmoid because we can classify binarily as a 0 or 1. In other words, we want the neural network to output a 0 or 1 to help us determine if a house is flooded or not. To train the model, we used 20 epochs with a batch size of 32. We wanted to do 20 epochs because we believed that for the amount of over 6,000 files we had to analyze, 20 full dataset iterations would be a fair chance for the neural network to learn. Similarly, with a batch size of 32, we could split the dataset into about 187 runs per epoch, which would allow the model to get better over time.

For the LeNet-5 model architecture, we decided to use 3 layers. We used 16 filters of size 3x3 with an average pooling size. We chose 16 because we thought it would be fair for the amount of different cases of damaged houses the model would see. Because damage varies between houses, with additional filters on top, there would be more than enough variance to get more accurate test results.

For the Alternate LeNet-5 model, we decided to use 4 feature maps with max 2D pooling per each map. We also ran this on 20 epochs. We decided to use max pooling to reduce the size of the output from the first layer we ran, which has 32 filters applied. To compile the model, we used a standard learning curve of 1e-4, still focused on a binary output to measure the accuracy of the model's performance.

Model Evaluation
Artificial Neural Network
The ANN model performed decently, yielding a test accuracy of 70.644% with a test loss of 0.56543. With a test accuracy score of 70%, this model is performing on the average, even though overall the model is fitting to the dataset 56% of the time. This implies that the model is underfit and it needs more epochs to get a better model accuracy, as we want to lower the test loss. It's predicting over half the time a

bad prediction, which we want to reduce. Overall, the ANN performed fairly, but we wanted to see higher accuracy from our model so we can be more confident in our results.

LeNet-5
Compared to the first model we tested, the LeNet-5 model performed significantly better, with a test accuracy of 88.745% with a test loss of 0.2912. The model had a higher accuracy in predicting the state of the houses, with a loss of less than 30%, meaning that the model made the correct prediction around 70% of the time. We were content with the results of the model and are significantly more confident in it predicting the correct prediction. We think we can do better in our model's performance accuracy, as we want to lower the loss error because around 30% of loss is still significant in autonomous model performance.

Alternative LeNet-5
The Alternative LeNet-5 performed the best out of all the models we created with a test accuracy of 96.459% with a test loss of 0.0864. This means that the model made the correct prediction over 96% of the time, a 0.08642 overall model accuracy. Additionally, the model made over around 91% of the time the correct prediction. Of all the models, we are the most confident in this one due to its high test accuracy and very low loss value. We believe that if we fed this model different test sets, it would be the most likely to make the correct prediction compared to the ANN and Lenet-5 models.

Model Deployment and Interface
To access our model you will need to be comfortable with Docker and Git.

Run the following command to pull the latest of our Github repository in a free directory:

```
git clone https://github.com/Costaki33/COE-379L-Projects.git
```

Move into the `project3` Directory

```
cd COE-379L-Projects/project3
ubuntu@costaki-coe379vm:~/COE-379L-Projects$ cd project3/
ubuntu@costaki-coe379vm:~/COE-379L-Projects/project3$ ls
Dockerfile  Project3.ipynb  api.py  data  models
```

Looking into the Dockerfile, change the 'costaki33' with your respective Docker user name. You will need to log into Docker Hub on your terminal to run the following command

```
ubuntu@costaki-coe379vm:~/COE-379L-Projects/project3$ cat Dockerfile
# Image: costaki33/project3

FROM python:3.11

RUN pip install tensorflow==2.15
RUN pip install Flask==3.0

COPY models ./models
COPY api.py ./api.py

CMD ["python3", "api.py"]
```

Then run the following to build and run the image. After running the image, create a new tab and log into your VM again.

```
docker build -t costaki33/project3 .
docker run -it --rm -p 5000:5000 costaki33/project3
```

You can run the following REQUEST commands. Based on the model's performance results, we decided to use the Alternate LeNet-5 to test our API calls. You can access Part 3 of the Jupyter Notebook to test this code.

```
import requests

# make the GET request:
rsp = requests.get("http://172.17.0.1:5000/model/info")

# print the json response
rsp.json()

{
    'description': 'Classifies images containing damaged and undamaged buildings
from Hurricane Harvey',
    'name': 'altlenet5',
    'version': 'v1'
}
```

```
import numpy as np
from PIL import Image

l =
np.array(Image.open('./data/split/test/damage/-93.528502_30.987438.jpeg')).tolist()

# make the POST request passing the sinlge test case as the `image` field:
rsp = requests.post("http://172.17.0.1:5000/model/predict", json={"image": l})

# print the json response
rsp.json()

{'result': [[0.0]]}
```