COE 379L: Project 3

Ashton Cole

April 11, 2024

1 Introduction

In this project, we build and test neural networks in Python, in order to predict hurricane damage on property from a satellite image. Three architectures are considered: a dense Artificial Neural Network, the LENET-5 Convolutional Neural Network, and a modified version of the LENET-5 model. The best-performing modified LENET-5 network is then saved to disk and developed into a containerized Flask *inference server*.

2 Methodology

2.1 Project Structure

1. `api.py`: Python code defining the endpoints of the Flask inference server.
2. `data/`: A folder used to hold training data.
   (a) `data_all_modified.zip`: An archive of the image dataset, i.e. 128 x 128 JPEG images, divided into `damage` and `no_damage` folders.
3. `dense-ann.ipynb`: A notebook building a densely connected Artificial Neural Network.
4. `Dockerfile`: A file with intructions to build a Docker image of the inference server from this directory.
5. `docker-compose.yml`: A file with instructions to build or pull the Docker image and then start a container running the inference server.
6. `docker-compose-arm64.yml`: An alternate `docker-compose` file for Mac ARM chip architecture, to resolve platform errors.
7. `lenet-5-cnn.ipynb`: A notebook building a LENET-5 Convolutional Neural Network.
8. `lenet-5-cnn-alternate.ipynb`: A notebook building an alternate LENET-5 Convolutional Neural Network described in https://arxiv.org/pdf/1807.01688.pdf.
9. `lenet-5-modified.keras`: A file holding the trained alternate LENET-5 network.
10. `preprocessing.ipynb`: A notebook splitting the unzipped `data_all_modified` images into training and testing folders.

2.2 Data Preparation

First, the data archive is unzipped and the `preprocessing.ipynb` notebook is run. This script creates separate directories for data used to train and independently test the models. Images are randomly compied into each, with 80% of the pictures being used to train the model. Images are labeled by the subdirectory in which they reside: `damage/` or `no_damage/`.

Before being applied to models, the images undergo additional processing steps. They are read-in using a TensorFlow Keras routine which returns a `Dataset` object. This class acts as a wrapper for the “independent” pixel RGB values, stored as a 3rd-order 128 x 128 x 3 tensor, and also the "dependent" class label. The tensor values are scaled down by a factor of $\frac{1}{255}$, so that they fall between 0 and 1. It also splits the training data into true training and validation subsets,

2.3 Model Construction and Training

All models are constructed using the TensorFlow Keras submodule. As simple Artificial and Convolutional Neural Networks, they are all sequential and are based on the same `Sequential` class. The differ, then, in the specific layers used, outlined in Table 1. Each accepts, as an input, a 4th-order tensor, i.e. a batch of n images, of dimension n x 128 x 128 x 3.

Table 1: Architecture of each neural network.

ANN		LENET-5		Modified LENET-5	
Layer	Parameters	Layer	Parameters	Layer	Parameters
Flatten	0	6 3x3 Convolutions	168	32 3x3 Convolutions	896
128 Dense	6,291,584	2x2 Average Pooling	0	2x2 Max Pooling	0
128 Dense	16,512	16 3x3 Convolutions	880	64 3x3 Convolutions	18,496
Dense	258	Flattening	0	2x2 Max Pooling	0
		100 Dense	5,953,700	128 3x3 Convolutions	73,856
		84 Dense	8,484	2x2 Max Pooling	0
		2 Dense	170	128 3x3 Convolutions	147,584
				2x2 Max Pooling	0
				Flattening	0
				10% Dropout	0
				512 Dense	2,359,808
				2 Dense	1,026
Total	6,308,354	Total	5,963,402	Total	2,601,666

The models are then fit to the scaled training and validation data, before being tested on the testing data. The compilation and fitting parameters are outlined in Table 2. The best-performing model is then saved as a file using TensorFlow's built-in functions.

Table 2: Parameters used to compile and fit each model.

Parameter	ANN	LENET-5	Modified LENET-5
Optimizer	`adam`	`RMSprop`	`RMSprop`
Loss	Sparse Cat. Crossentropy	Sparse Cat. Crossentropy	Sparse Cat. Crossentropy
Epochs	5	5	5
Batch Size	32	32	32

2.4 Containerized Inference Server

A simple Flask server is built in Python, which reads in the saved model and uses it to predict on images. It contains the following routes.

- `models/hurricane-damage/v1`
  - `GET`: Returns basic metadata about the inference server.
  - `POST`: Classifies a satellite image of property as having been damaged or having not been damaged by a hurricane. This route requires a byte stream of a 128 x 128 JPEG image, wrapped in a dictionary under the key `"image"`. It returns a JSON dictionary in the following form. `{'result': [probability_damage, probability_no_damage]}`

The API is then put into a Docker image with the model file. The image may be found on Docker Hub as `ashtonvcole/hurricane-damage-api`, or built manually. It may be run with a binding to port 5000, which is where Flask expects incoming HTTP requests. A `docker-compose.yml` file is also included in the project repository to streamline building and deployment with the `docker-compose` command.

It is worth noting that as of now, the Docker throws errors when deployed on new Macs with the ARM architecture. However, the container should deploy as expected on any x86-architecture computer.

With the container deployed, the API may be tested on the local host. Requests may be conveniently made through a Python interpreter or script like the one below.

```
import requests

url = 'http://127.0.0.1/models/hurricane-damage/v1'
file = open('image.jpeg', 'rb')
files = {'image': file.read()}

response = requests.post(url, files=files)
print(response.json())
```

3 Results

Table 3: Results for each model.

Metric	ANN	LENET-5	Modified LENET-5
Training Accuracy	0.6637	0.8556	0.9320
Validation Accuracy	0.6681	0.8461	0.9455
Testing Accuracy	0.6644	0.8415	0.9395

4 Conclusions

As seen in Table 3, the Modified LENET-5 model performed the best of the three. Interestingly, it is also the most efficient, containing the fewest trainable parameters. With a roughly 94% accuracy, the user may trust it to reliably classify most images. Thus, it is selected for the inference server.

At a minimum, with similar values for training, validation, and test accuracy, none of the models show overfitting. This is a sign, however, that in the future, additional experimentation could be done to increase accuracy. Other combinations of layers could be tested, or more training epochs could be run for each model, up to the point of maximum accuracy without overfitting.

Regardless, the results of these models show the powerful ability of machine learning to automate away tedious tasks like inspecting individual properties for damage after a hurricane. Such models could be used to quickly identify zones of damage, to better allocate recovery resources and prepare regions for the future.