COE 379L

Project 3

Jason Kim jk46965

Braulio Lopez bl27466

Introduction

The following report outlines the task given for the course Software Design For Responsible Intelligent Systems (COE 379L). The third project of the course focuses on building multiple neural networks based on different architectures to classify images. Specifically, this project uses a dataset containing satellite images of damage and undamaged buildings from Texas after Hurricane Harvey. The Purpose of this project is to use this dataset to build the neural network to classify between damage and undamaged. The Project evaluates each of the networks developed and chooses the best network to deploy.

Data Preparation

In order to make this project possible, it was necessary to obtain the dataset from the Github provided in the project description and get it in the virtual machine. After successfully downloading the dataset, data processing is a necessary step to make our models. Therefore, it's essential to make sure that train and test directories are empty to start. To do this, we first have to import “shutil” to remove the directories and their contents. After doing that we then create the train and test directories for damage and no damage category. Moreover, we collected all paths for images for damage and no damage, and we split them into train and test sets in a 80:20 ratio, respectively at random using the *random* library.

Furthermore, it is also important to ensure there are no overlaps between the two splits. If we had overlaps this would cause some performance metrics such as the accuracy to be inflated.

After successfully getting our train images in the training folder, we need to preprocess the images for training the models.

For that we successfully got the image sizes, by implementing a python code that prints the number of images along with its dimensions “7152 images with dimensions 128x128” - we referred to ChatGPT for help on this part. We did that because we have to select a target size for each image (128,128,3), so the model can be trained on them. Rescaling was another part of the processing; we used Rescaling(scale=1./255) to rescale pixel values from the typical range of [0, 255] to the range [0, 1].

Model Design

After successfully preparing the data, we are ready to build the neural networks. For this we use 3 distinct architectures: ANN, Lenet-5 CNN, and an Alternate-Lenet-5 CNN architecture from an outside research paper.

For the first model, ANN, the input layer had to be flattened and we used two hidden layers with 512 and 256 neurons with ReLU activations.

For the Lenet-5 CNN, two sets of convolutional and pooling layers for extraction were used. After that, we implemented two layers of 120 and 84 neurons with ReLU, along with a softmax layer for binary output.

Finally, with the alternate LeNet-5, we improved feature extraction by adding convolutional layers and max pooling. To reduce overfitting, we used dropout layers and L2 regularization in the last dense layer.

Mode Evaluation

From the models we can see that the Alternate-Lenet-5 CNN architecture performed the best. We are *extremely* confident in our chosen model as per the high accuracy and low loss rate,

and due to its higher model capacity (has the highest number of parameters relative to the other models). It's expected for a more complex model to capture more intricate patterns for images, which explains why the Alternate-Lenet-5 model performed best. Furthermore, we accounted for the possibility of overfitting, especially with such a high parameter number, and addressed it by implementing max-pooling layers and dropout regularization, thereby maximizing the training and performance of our chosen model.

Model	Loss	Accuracy
ANN	0.6381585597991943	0.6644783020019531
Lenet-5-CNN	0.322668194770813	0.8590855598449707
Alternate-Lenet-5	0.11340105533599854	0.9716295599937439

Model Deployment and Inference

Assuming the user has forked or pulled the latest changes from our repository, in order to deploy our model, they must first ensure they are in the Project3/ root directory where the docker-compose.yml and Dockerfile are located and run the following command(s):

```
$ docker-compose up
You may add '-d' to put it in daemon mode.

$ docker ps -a
To ensure that the container is up and running.
```

In doing so, the model should be successfully deployed and running inside a docker container. Here are the possible endpoints to hit from the Flask app.py:

GET '/model_summary'

- returns info about model (layers_count, parameters_count, input_shape, output_shape)

GET '/model_summary_table'

- returns summary of model (model.summary()) in tabular format.

POST '/classify_image'

- expects binary message payload containing image (.jpeg).
- returns results of the inference in JSON.

From the command line, please run the following commands, respectively, to call endpoints:

```
$ curl localhost:5000/model_summary

$ curl localhost:5000/model_summary_table

$ curl -X POST -F "image=@/path/to/your/image.jpeg" localhost:5000/classify_image
```

The GET endpoints that return info and summary about the model essentially details the architecture of the model. The user will be able to see some metadata, the layers, and parameters count, where more layers and higher parameters mean the model is more complex.

For the POST '/classify_image' that predicts a single sample image, the user should expect an output something like:

```
{
  "prediction": "Damaged",
  "probability_no_damage": 3.882407327182591e-05
}
```

The "prediction" is based on the "probability_no_damage". The probability value is an output when calling model.summary(), which returns a value that represents the probability of it being class 0, which in our case would be no_damage. Thus, the backend logic takes this probability and if it's <0.5, we can infer that the image passed in has no damaged buildings, vice versa. In the specific example, we would interpret it as the image having a probability of 0.000039 of having no damaged buildings, hence the image being classified as containing damage.

Refer to the github repo to see comprehensive examples of the curl commands and expected output.

The github repository can be accessed at:

https://github.com/jasonthekim/COE379L-Projects/tree/main

ChatGPT Reference

To be frank, we were rusty with Flask and referred to ChatGPT for a lot of help in creating our Dockerfile, docker-compose.yml, and endpoints.

References

Cao, Q. D., & Choe, Y. (n.d.). *Building Damage Annotation on Post-Hurricane Satellite Imagery Based on Convolutional Neural Networks*. https://arxiv.org/pdf/1807.01688.pdf

Stubbs, J., & Jamthe, A. (n.d.). COE 379L: Software Design For Responsible Intelligent System. https://coe-379l-sp24.readthedocs.io/en/latest/index.html