COE 379L: Software Design For Responsibly Intelligent Systems

Performance Evaluation of CNN and ANN Models on Building Damage Prediction

Pranjal Adhikari and Kelechi Emeruwa

April 11, 2024

Table of Contents

1 Introduction

This project focuses on working with a data set (found here) consisting of satellite images captured after Hurricane Harvey made landfall in the Texas region. The images are organized into two categories: damage and no-damage. Each category is composed of images which include buildings and structures in a certain area. The objective is to develop neural networks from different model architectures to assess and classify these images into the correct category. As this task is centered around binary classification, the models are required to only distinguish between the two categories of damage.

Various different model architectures can be used to develop neural networks. The model architectures utilized specifically in this project include Artificial Neural Network (ANN), Lenet-5 Convolution Neural Network (CNN), and Alternate-Lenet-5 CNN.

1.1. Data Preparation

Before developing any of the models, preliminary information about the data set was investigated to prepare the data to be used in the model design. Statistics, such as the total number of images in each category, the size and format of each image file, and the dimensions of each file were sought out. The data set includes ~21,000 total image files with roughly two-third of them in the damage category, and the other part in no-damage. Furthermore, the average size of the images in the damage category is 2.5 kB, and 3 kB in the no-damage category, with all in color and .jpg format. Lastly, all images within the data set are 128x128 pixels, which was an important statistic to gather and enforce when developing the models.

For the ANN model, the images were first flattened to generate a 1D vector to implement into the first dense layer of the neural network. Next, the image pixels were normalized to improve the convergence of the models during the training phase. In this project, the pixels were normalized by a value of 255. Afterwards, the images in each category were split into train and test data sets, with 80% separated to train and 20% to test. As the model is being developed to predict whether an image falls under two

categories, the target variable (y variable) was converted to one-hot encoding using the function *keras.util* from the *TensorFlow* library.

For the Lenet-5 and Alternate-Lenet-5 CNN models, the images were kept as 2D arrays and similar to the ANN model, the images were split into train and test data sets with the 80/20 split. The *Rescaling* function from *keras* was used then to rescale the images' pixel values within the range 0 to 1, which allows the neural network to be trained in an accurate manner.

2. Model Design

While creating a neural network, the architecture requires implementation of different components, such as the type of model, the number of layers, and the number of perceptrons. The architectures in this project are all a Sequential model type, which include a linear stack of layers and use *Keras.model* for creation.

ANN Model

Our first model uses a zero-biased input layer followed by two pairs of zero-biased. Wherein, the first layer in the pair uses a ReLU activation function with 256 perceptrons and the second uses a linear activation function with 128 perceptrons. This approach was adopted based on research showing how zero-biased layer pairs can increase the rate of convergence and accuracy of an ANN model on image classification problems (Konda et al, 2016). Next, the input layer utilizes the dimension of the pixels in the images (128x128) for the input shape, with 784 perceptrons and the activation function 'relu.' Finally, a 'softmax' activation function is added in the output layer, as the problem includes classification and ensures that the sum of the probabilities for images belonging to a category is equal to 1.

Lenet-5 CNN

The Lenet-5 model includes two alternating convolution and pooling layers, two fully connected layers, and an output layer. The first convolution layer includes 6 filters, with size 3x3, the input shape of the pixels in the images, and activation function 'relu.'

The next convolution layer is similar, but with 16 filters. Two connected layers follow with 120 and 84 perceptrons, and finally an output layer with 2 perceptrons (for classifying into two categories) and the activation function ‘softmax’.

Alternate-Lenet-5 CNN

The Alternate-Lenet-5 model is similar, but with 4 alternating convolution and pooling layers. The first layer begins with 32 filters, increases to 64 in the second layer, with the third and fourth layers having 128 filters each. All three utilize the ‘relu’ activation function. A dropout layer is added at a rate of 0.5 to reduce overfitting of the model, followed by a dense layer with 512 perceptrons. An output layer with 2 perceptrons is added lastly.

3. Evaluation

Accuracy scores were calculated for each of the three models with the test categorical data. The ANN model scored an accuracy of 0.77, the Lenet-5 model with 0.84, and the Alternate Lenet-5 model with 0.98. Thus, it can be concluded that the Alternate Lenet-5 model performed the best out of the three. In terms of utilizing the Alternate Lenet-5 model for image assessing tasks, we are confident in the ability to do so.

4. Deployment

The model is served over an inference server in two ways. The first is by pulling the containerized application from an existing Docker image, kelach/lenet5a_model:1.0. Pulling the existing Docker image requires users to simply clone the repository, cd into the directory on their terminal, and run the command *docker-compose up*. The second is to build the image using the *docker-compose.yaml* file. In addition to cloning and cd-ing into the repository locally, users will also need to build the model locally as it is not included in the repository. However, users can easily build the model using the

interactive Python notebook provided in the repository (although they will need to change the directory path the notebook points to).

Once the model is running, users can make requests using the curl command, or via a dedicated inference runner python file. The inference runner file is easier to use, as it automatically converts all .jpeg, .png, or .jpg data found in the "data" folder of the repository into JSON format. Otherwise, this conversion is left for the user to fulfill. Additional configuration details are outlined in the README.md file of the repository which can be found here.

5. Resources

- https://coe-379l-sp24.readthedocs.io/en/latest/index.html
- https://www.asimovinstitute.org/neural-network-zoo/
- https://openreview.net/pdf/1WvovwjA7UMnPB1oinBL.pdf