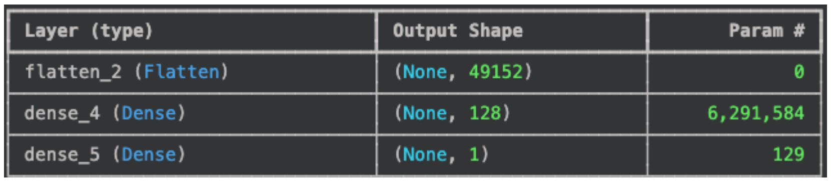
COE379L - Project 03 Report
Martin Infante
mbi244
Slip days used: 5

Data preparation

To begin, the data preparation process involved pulling datasets for damage and non_damage images into my local repository. I then made new directories for *train* and *test* and split the image paths into 80% train and 20% test. In the train set there were 11336 damage images, and 5721 no_damage images. In the test set there were 2834 damage images and 1431 no_damaged images. An important observation from this dataset is that there is nearly double the amount of images for damaged vs undamaged. Since we are conducting binary classification, it is important to note then that the model may be skewed when conducting tests in the general case. Then, I investigated the datasets to determine basic attributes of the images. For the first 10 files in each of the following directories: *train/damage. train/no_damage, test/damage,* and *test/no_damage,* I outputted the file size, the image dimensions and its image mode. Output showed that images are 128x128 pixels with an image mode of RGB. Furthermore, I printed the images to console to do a brief visual overview of the dataset. Afterwards, I preprocessed the data by using *keras.utilis.image_dataset_from_directrory* passing input parameters to the image size, batch size, validation split and seed number for reproducibility. A validation split of 0.2 was used, and *subset="both"* was used for the train data. Test data was modified similarly, except the *subset* was not specified because we do not want the test set to split. Both datasets were also rescaled by dividing by 255.

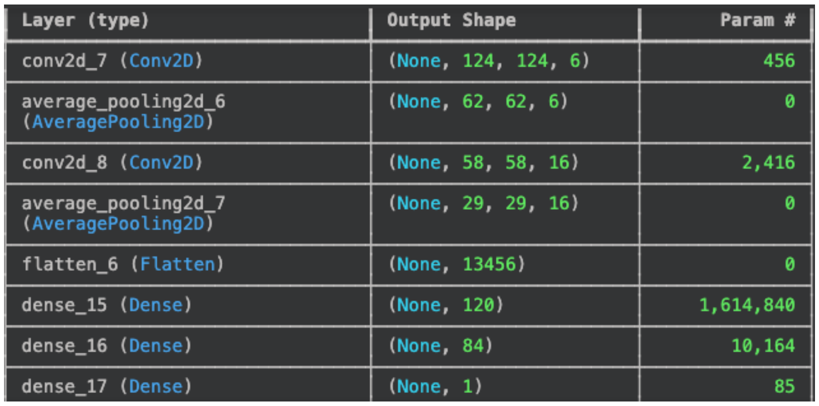
Model design

Three different model architectures were explored: a dense ANN, The Lenet-5 CNN architecture and an Alternate-Lenet-5 CNN architecture. All models were compiled with *adam* optimizer, optimizing the *binary_crossentropy* loss function, making improvements to *accuracy*. All models were also fit to the *train_rescale_ds* with validation_data=*val_rescale_ds.* Now, the architecture of ANN model consisted of a flattened input layer of size 49152 (because images are 128x128 RGB), followed by one dense hidden layer with 128 perceptrons. The activation function used here was *relu.* The output layer was one perceptron with the *sigmoid* activation function. In total, there was 6,291,713 total parameters. The model was initially tested with 100 epochs, but post-analysis showed that accuracy on validation set peaked around the 52nd epoch, at a value of .7877. The model accuracy decreased to .744 on the test dataset. A summary of the model is depicted below:



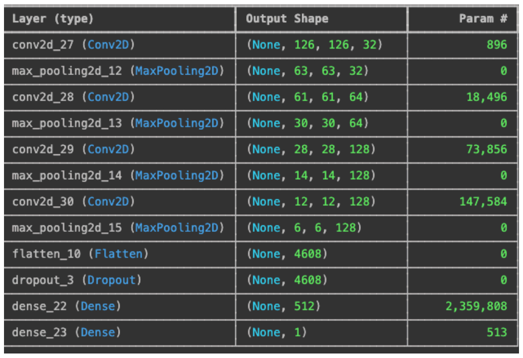
Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 49152)	0
dense_4 (Dense)	(None, 128)	6,291,584
dense_5 (Dense)	(None, 1)	129

The next model, Lenet-5 CNN architecture, involved two initial Convolutional-to-Pooling layers, with kernel size of 5 x 5. Pooling size used was 2 x 2. The feature maps were then flattened to feed into the dense layers, consisting of one with 120 perceptrons, and the next with 84 perceptrons. The output layer consisted of one perceptron with the sigmoid activation function. Model was initially tested with 50 epochs, but showed marginal increases after the 21st epoch, with a validation accuracy value of .891. Surprisingly, the *lenet_model* had an accuracy of 0.896 on the test data. A summary of the model is depicted below:



Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 124, 124, 6)	456
average_pooling2d_6 (AveragePooling2D)	(None, 62, 62, 6)	0
conv2d_8 (Conv2D)	(None, 58, 58, 16)	2,416
average_pooling2d_7 (AveragePooling2D)	(None, 29, 29, 16)	0
flatten_6 (Flatten)	(None, 13456)	0
dense_15 (Dense)	(None, 120)	1,614,840
dense_16 (Dense)	(None, 84)	10,164
dense_17 (Dense)	(None, 1)	85

The last model, Alternate-Lenet-5 CNN architecture was the most elaborative, including more convolutional layers, a dropout layer, and a dense layer with 512 perceptrons. The activation functions in all layers was *relu,* except the output layer used *sigmoid.* Due to time constraints, the model was only evaluated with 20 epochs, but achieved a peak validation accuracy score of .9804 on the 19th epoch. When evaluated on the test data, the model achieved an accuracy score of .966. A summary of the model is depicted below:



Layer (type)	Output Shape	Param #
conv2d_27 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_12 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_28 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_13 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_29 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_14 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_30 (Conv2D)	(None, 12, 12, 128)	147,584
max_pooling2d_15 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten_10 (Flatten)	(None, 4608)	0
dropout_3 (Dropout)	(None, 4608)	0
dense_22 (Dense)	(None, 512)	2,359,808
dense_23 (Dense)	(None, 1)	513

Model evaluation

Among the three models, the Alternate-Lenet-5 CNN demonstrated the best performance on the test dataset, achieving the highest accuracy score of 96.6%. This suggests that architecture's inclusion of more convolutional layers with a steady increase in feature map size helped achieve better results.

Model deployment and inference:

To deploy our model, make sure all required files–including the Dockerfile, docker-compose.yml, api.py, the /models directory with alt_lenet.keras, and the /data directory–are present in your local repository. To start the inference server run *docker-compose up*, which will launch the server on port 5000, utilizing our trained Alternate-Lenet-5 CNN model. The server handles two API endpoints: a GET request to /summary, which returns model metadata in JSON format, and a POST request to /inference, which accepts an image file (sent by the user in the "image" field of the flask.request.files structure) and returns a prediction of either "damage" or "no_damage". For example once the docker container is running, you can test the server by running curl localhost:5000/summary on a separate terminal to receive a description of the metadata of the model.