# Project 3 Report

By Steven Oh and Trey Gower

## Preparing the Data

To prepare the data, 80 percent of the total data was used for training and the rest of the data (20 percent) was used for testing the model. The data was split and moved into a new directory to use later for training and testing.

To preprocess the data, the `tf.keras.utils.image_dataset_from_directory()` functionality was used. This function converts image files in a directory into a TensorFlow `tf.data.Dataset`, which will be used to train and test the models created later on.

## Model design

There were three architectures that were explored during this project: the dense Artificial Neural Network (ANN) model, the LeNet-5 Convolutional Neural Network (CNN) model, and the Alternate-LeNet-5 CNN model described in the "Building Damage Annotation on Post-Hurricane Satellite Imagery Based on Convolution Neural Networks" article.

### Dense ANN:

For the dense ANN model, we decided to flatten the model at the beginning in order to convert the tensor to a form that can be used with the Dense ANN model. Then, we have an initial layer with 120 perceptrons, a second layer with 64 perceptrons, and lastly the final layer with 1 perceptron. This last layer uses the activation `sigmoid` to help with binary classification. This is relevant for this project as there are two categories that the model needs to sort (damaged or not damaged). Then, the `adam` optimizer (used to minimize the loss function while training) was used along with the `binary_crossentropy` loss (works well with sigmoid).

### LeNet-5 CNN:

For the LeNet-5 CNN model, we included two pooling and two convolutional layers. The first convolutional layer contained 6 filters, a kernal size of 3x3 and an image input of 150x150 in black and white. The second convolutional layer had a 16 filters and even better filter the damages in a building. Now for what makes a CNN model what it is, the pooling layers.

For both pooling layers, we decided to use average pooling instead of max pooling. This decision was based on the images we were having the model analyze. Because we are classifying damaged buildings, it is much better to use average pooling as it takes a whole region of the picture into account versus the maximum pixels in a cell. With max pooling our classifiers would be much less accurate.

When summarizing the metrics of the LeNet-5 model we look at computational cost, accuracy, and validation accuracy. Comparatively to the dense ANN model and the Research Model Below, the wall clock time for each epoch was significantly faster averaging around 35s per. Interestingly, we also see the accuracy is higher sitting at 89%. This suggests that the model was overfitting the data. And, when looking at validation accuracy we can see this is verified as we visually see lower validation scores for the model.

### Research Suggested Model:

As suggested by a research paper given to us we tried an Alternate-LeNet-5 CNN model. The difference is that we included 2 extra pooling and convolutional layers for extra filtering. And, instead of average pooling we used max pooling. As stated above, I still believe the max pooling will end up causing less accuracy, which can be somewhat seen by the accuracy score from the model at 69%. However, we can now see that this model is not overfitting with higher validation scores around 95%. Plus, the wall clock time per epoch is not substantial, averaging 150s.

## Model evaluation

Only one model had a good accuracy score compared to the other two; the LeNet-5 CNN model had an accuracy of about 0.90. The Dense ANN model and Alternate-LeNet-5 CNN model didn't do as poorly, but much lower than the LeNet-5 CNN with an accuracy of 0.75 and 0.69, respectively. Therefore, the LeNet-5 CNN architecture had the best performance amongst the other two. With an accuracy score of 0.90, there is much confidence that the LeNet-5 CNN model will perform well with new images. Especially with the model being a CNN model, there is much confidence as CNN structures use the convolution mathematical operation to specifically process data like images. To put it simply, even if the image is cropped, rotated, or translated (which in this case can be very common), the LeNet-5 CNN model will have pretty good results.

## Model deployment and inference

To deploy/serve the model, one must pull down the GitHub repository and use the `docker-compose up` functionality in the command line. The docker pull command (`docker pull jaeestee/ml-buildings-api:latest`) is not required as `docker-compose up` should automatically pull it down if needed (however pull down manually if it is causing issues). A very important step before running the `docker-compose up` command is to edit the `docker-compose.yml` file and make sure to add a volume mount to the files that you wish to predict with the model. An example is shown in the yml file itself and further explanation can be found in the `README.md`. As soon as `docker-compose up` is ran, the server should be up and running.

There are two ways to use the model for inference. The first is through the command line and the second method is to use a Python script. For the first method with the command line, the user can use the curl command to send an HTTP request to the server. For the second method, the user would use the requests library to create HTTP requests. More in-depth information and examples can be found in the `README.md` file inside the Project_3 directory.