# Post-Hurricane Structure Damage Assessment Leveraging Aerial Imagery and Convolutional Neural Networks

Allan Kapoor[1]

## Introduction

This report summarizes an effort to automate structure damage assessments based on post-hurricane aerial imagery. A labelled training dataset of images of structures from the Houston area just after Hurricane Harvey was acquired from the University of Washington Disaster Data Science Lab. Several neural network architectures were trained and evaluated, from basic architectures to transfer learning models with pretrained weights. Models were trained on Google Public Cloud virtual machines leveraging multiple GPUs to enable fast and efficient iteration. The final model achieves an accuracy of 0.9775 on test data. This model could be used by local, state, and federal natural disaster responders to quickly develop damage assessment maps based on a single fly-over by imaging aircraft after a hurricane.

## Problem Statement

Natural disasters, especially hurricanes and extreme storms, cause widespread destruction and loss of life and cost the U.S billions of dollars each year. Of the 258 U.S. weather disasters since 1980, tropical cyclones (hurricanes) have caused the most damage: $945.9 billion total, with an average cost of almost $21.5 billion per event[2]. Extreme storms have caused widespread damage to major cities such as Hurricane Katrina (New Orleans, 2005), Superstorm Sandy (New York City, 2012), and Hurricane Harvey (Houston, 2017). More extreme weather due to climate change has increased frequency and intensity of extreme storms - 2020 was the sixth consecutive year in which 10 or more billion-dollar disaster events occurred in the U.S[3].

Effective response after a natural disaster strikes is critical to reducing harm and facilitating long term recovery. Damage assessments in the hours/days after a disaster can help efficiently allocate immediate response resources and determine spatial patterns of damage. Damage assessments also help calculate economic impact estimates which are necessary to secure and approve recovery funding.

[1] Thanks to Shmuel Naaman for mentorship and advice on feature engineering/algorithms

[2] https://coast.noaa.gov/states/fast-facts/hurricane-costs.html

[3] ibid.

However, completing fast and accurate damage assessments immediately after a disaster faces several challenges. Access to damaged areas may be physically cut off due to impacts to the transportation network. An extreme storm may impact hundreds of thousands of structures across hundreds of square miles, requiring significant resources if the assessment is done manually. Especially immediately after an event, limited resources may be tied up with immediate rescue and triage. *A model that can perform automated damage assessment based on remote sensing imagery quickly captured by a few planes flying over a disaster site would greatly reduce the effort and increase speed in which useful data (location and extent of damaged structures) could be put in the hands of responders.* Entities that would benefit from this capability include responders from the federal to local level such as the Federal Emergency Management Agency (FEMA), the Department of Defense (National Guard, etc.), state offices of emergency services, and county/municipal governments.

> **Problem Statement:** In order to support effective natural disaster response and recovery, how can we leverage remote sensing imagery to quickly and efficiently produce accurate damage assessments after extreme storms?

## Data Source

Data for this project was sourced from a labeled training dataset[4] made publicly available by Quoc Dung Cao and Youngjun Choe at the University of Washington's Disaster Data Science Lab. The dataset consists of images of structures cropped from aerial imagery collected in the days after Hurricane Harvey in Houston and other nearby cities. Images are tagged based on whether the structure depicted suffered damage during the hurricane or not.

Images in the dataset are organized into training, validation, and test sets. The training dataset includes 10,000 images (5,000 no damage, 5,000 with damage) and the validation and test sets each include 2,000 images (1,000 no damage, 1,000 with damage). The filename for each image includes the latitude and longitude coordinates which can be used to plot the locations of the structures in relation to each other (see below).

The data set is very organized and did not need much cleaning or wrangling. Prior to EDA, I did scan all images to verify that none were corrupted. Due to the relatively small number of images in the training dataset, image augmentation was included in the model pipelines (see Modelling below).

[4] Cao, Quoc Dung and Choe, Youngjun. 2018. University of Washington Disaster Data Science Lab. Accessed: https://ieee-dataport.org/open-access/detecting-damaged-buildings-post-hurricane-satellite-imagery-based-customized

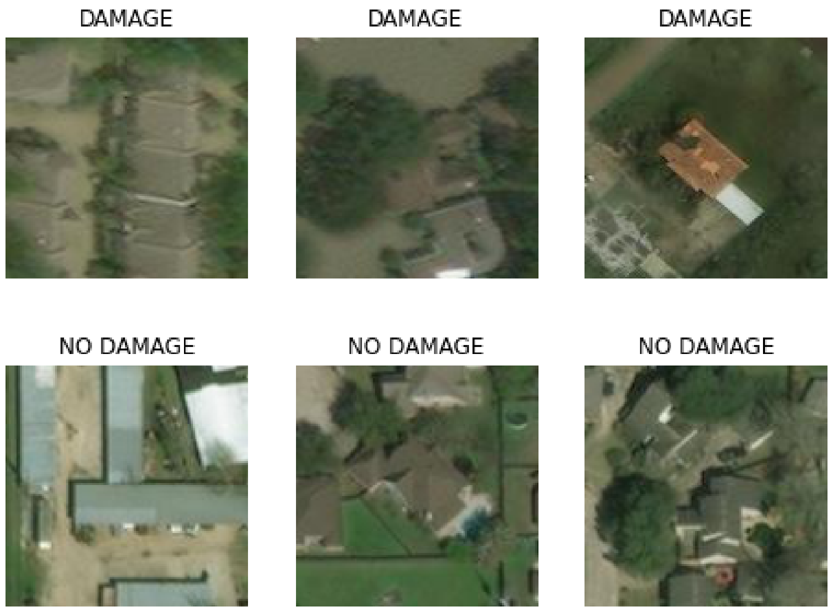
## Exploratory Data Analysis[5]

Before attempting to train a neural network, it was important to perform some initial data exploration to understand patterns in the data. First I plotted example images from each class in order to get a visual understanding of the dataset. Then I explored trends in the data by plotting various summary images - visual representations of the mean, standard deviation, and contrast between the two classes. Next I plotted eigenimages based on principal component analysis (PCA) which reveal common features in the images. Finally, I plotted the geographic locations of the images in order to understand the spatial distribution of the structures in the dataset.

### Visual Inspection

The images are color, in RGB format. The figure below compares 3 randomly selected damage images to 3 randomly selected no damage images. From viewing these examples along with others, some patterns emerged:

- Many (but not all) images have flood waters surrounding the structures, so there is a difference in ground texture/color.
- Other damage images have small objects strewn across the ground. However, many images of no damage also display this visual pattern.
- Images of structures with no damage seem more likely to feature visible ground or pools with blue-ish water (uncovered by flood waters).
- In many cases it isn't obvious to the human eye if an image shows damage or no damage.



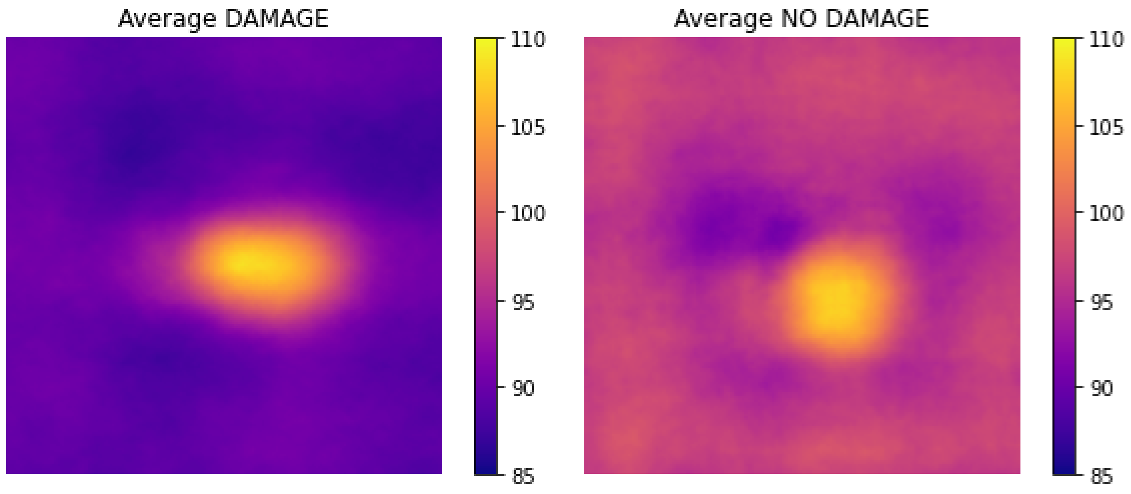
[5] Notebook: https://github.com/allankapoor/wildfire_prediction/blob/master/Step1_EDA.ipynb

## Summary Images

In classification problems based on structured (tabular) data, it is common to explore summary statistics for each feature (column). For this image classification problem I used the numeric RGB color values in the images to calculate summary statistics by pixel (feature) and then plot them visually. By plotting visual summaries of the two classes differences can be identified between them that could inform decisions about neural network architecture.

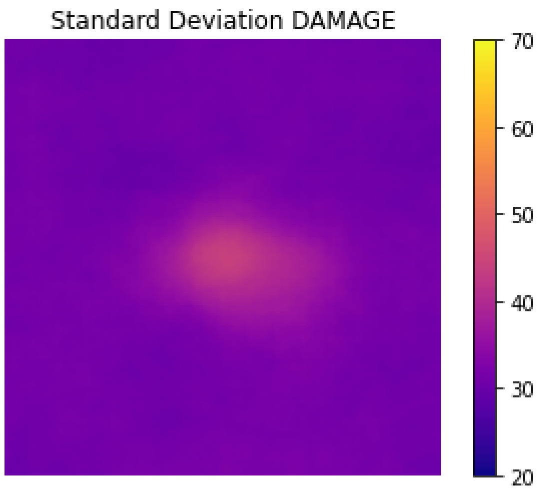
**Mean Value by Class**

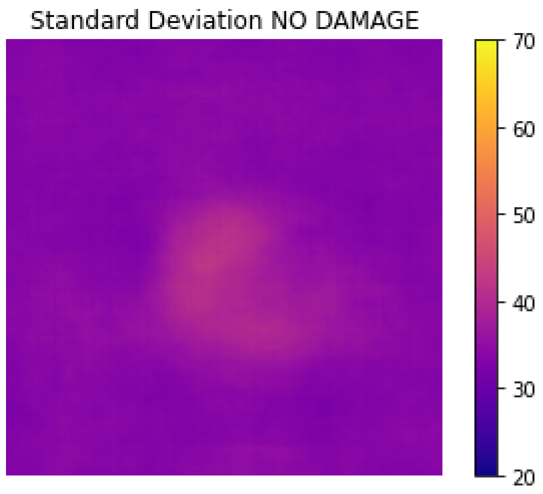
The figures below depict each pixel's mean value across all images by class. For both damage and no damage images we see that a structure tends to be located within the center of the image. For damage images, the pixel values around the structure tend to be lower in value than around the no damage images. Perhaps this is because damage images tend to have flood waters around the structures, which are a different color than unflooded ground.



**Standard Deviation by Class**

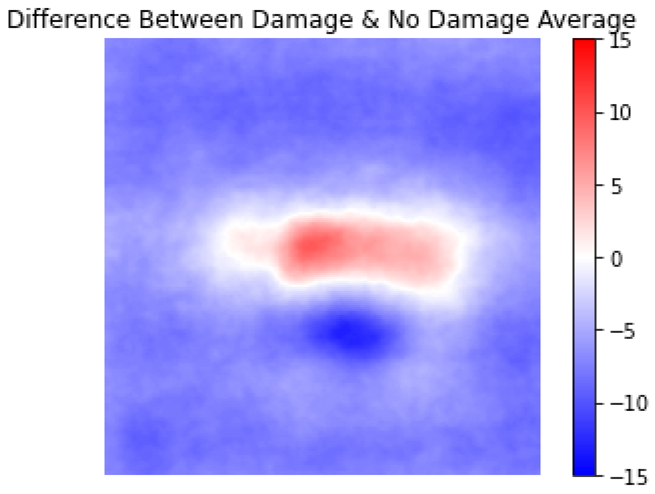
The figures below are similar to those above but instead of depicting the mean pixel value they depict the standard deviation for each pixel by class. Standard deviation around the edges of the image appears to be (slightly) greater for the no damage images. Perhaps this is because visible ground around the structures creates more variation between images in that class.





**Contrast Between Mean Value by Class**
In the figure below, pixel values are the difference between the mean images for the two classes (damage and no damage) displayed above. Dark blue and dark red pixels are where there is the greatest difference between mean values across the two classes and white is where there is the least.

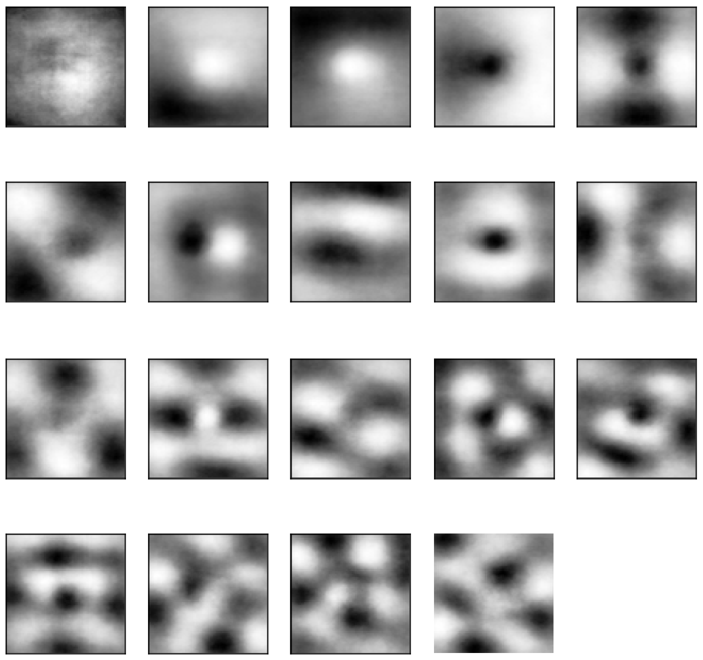
The white pixels appear to form the outline of a structure - this is expected since all images of both classes have a structure in the center of the image. The difference values are higher in the center and edges (area surrounding the structure).



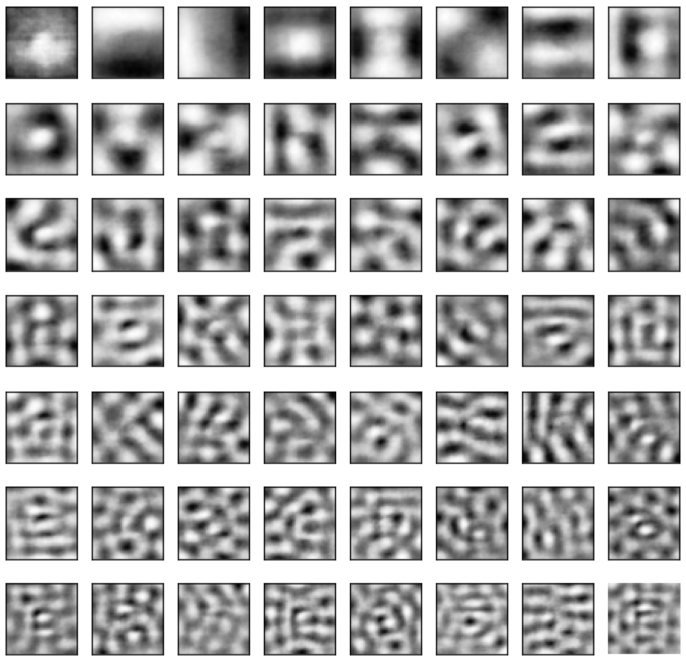
## PCA + Eigenimages

The functions below leverage PCA to produce eigenimages for the two image classes. The eigenimages are not easily recognizable to the human eye, unlike eignimages based on faces or other images with more regular repeating features. There are hints of basic geometric shapes with right angles, as would be expected from images of structures.

For images depicting structures with damage, 19 principal components explained 70% of the variation:



For images depicting structures with no damage, 56 principal components explained 70% of the variation:



The eigenimages are not easily recognizable to the human eye, unlike eigenimages based on faces or other images with more regular repeating features. There are hints of basic geometric shapes with right angles, as would be expected from images of structures.

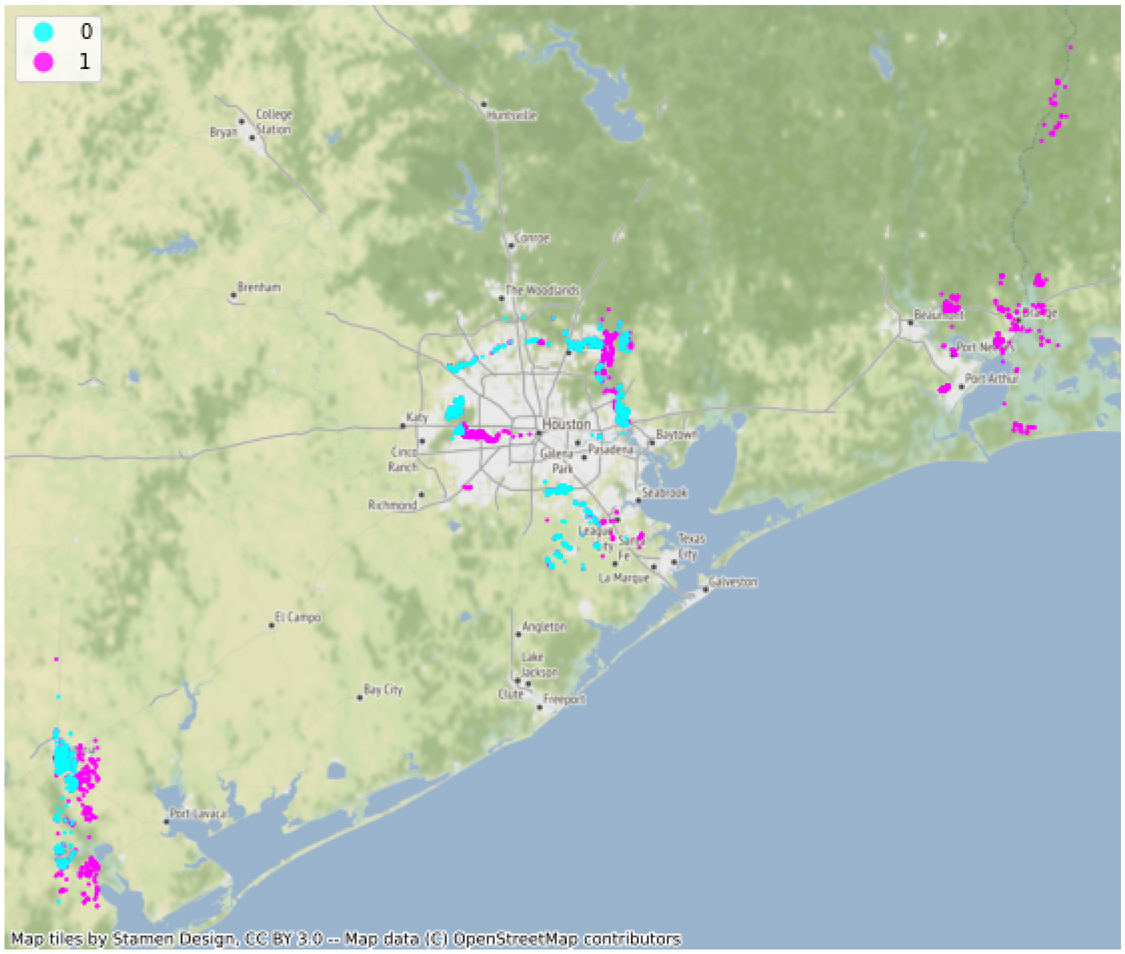
It is worth noting that in order to explain 70% of the variation in each class, the damage class only requires 19 principle components while the no damage class requires 56. This is consistent with our earlier finding that the standard deviation for pixel values across the no damage class tended to be higher than the with damage class.

## Geographic Distribution

Finally, I investigated the geographic distribution of the training data by class. This is possible because the GPS coordinates are contained in the filename for each image file. The figure below depicts the locations of each image in the training dataset and whether it is damaged or not damaged.

We can see that the training data appears to come from 3 distinct areas. In one area, all structures in the training dataset are damaged, while in the other two there is a mix. Within those two there are clear spatial patterns of where flooding/damage occurred and where it didn't. This is not surprising given that the hurricane (and flooding) impacted areas differently based on topography, urban form, etc. However, it does also indicate that the examples from each class often come from entirely different areas. There is a danger that when training the neural networks models, they may learn to identify differences that are more to due with differences in appearance of structures between those areas, rather than differences that are actually due to hurricane damage.

Point locations of Training Images, Damaged or Not



## Modeling[6]

### Modeling Overview

This section describes the general approach used to train and iterate on neural network models. All models were developed with TensorFlow using the Keras API. While the data exploration described above was carried out on a desktop computer, the models were trained on a Google Cloud virtual machine with a GPU[7], enabling fast training and iteration.

Model performance was evaluated using accuracy on the validation set (unused for training). This is an appropriate metric because there are an equal number of instances of each class. I trained each model for 50 epochs with an early stopping rule - if the model did not improve after 10 epochs, the training would end and the model weights from the epoch with the highest

[6] Notebook: https://github.com/allankapoor/wildfire_prediction/blob/master/Step3_Modeling.ipynb
[7] Virtual machine specifications: 8 vCPUs (30 GB RAM) + 1 NVIDIA T4 GPU (16 GB)

validation accuracy score would be saved. In all cases, training ended due to early stopping before 50 epochs.

Given that this is an image classification problem, it was assumed that a convolutional neural network architecture will perform best. Starting with a very simple model featuring three convolution layers and three dense layers, I gradually added layers until I no longer saw gains in accuracy. Then I experimented with altering hyperparameters, with the goal of paring back model complexity (and training time) while maintaining performance and improving convergence.

## Baseline Model

I first created a simple baseline model with three convolution layers, three dense layers, and an output layer with 2 nodes (corresponding to the two classes). Details of the model architecture are summarized in the table below. Some initial modeling decisions were made here and carry through to the other models:

- The ReLU activation function is used for all layers except the final output layer because it is computationally efficient and known to lead good results in most cases[8]
- Batch normalization was included after the convolution layers but before activation based on experimentation (when batch normalization came after activation, validation accuracy dropped substantially).
- Adam optimizer was used for updating network weights because it is known to perform well on image classification problems[9]. Adam adjusts the learning rate over time.

| **Layer** | **Output Shape** | **# of Params** |
|---|---|---|
| Rescaling | (128, 128, 3) | 0 |
| Convolution (filters=32, kernel_size=5, strides=2) | (64, 64, 32) | 2,432 |
| Batch Normalization | (64, 64, 32) | 128 |
| Activation (ReLU) | (64, 64, 32) | 0 |
| Convolution (filters=32, kernel_size=3, strides=1) | (64, 64, 64) | 18,496 |
| Batch Normalization | (64, 64, 64) | 256 |
| Activation (ReLU) | (64, 64, 64) | 0 |
| Convolution (filters=32, kernel_size=3, strides=1) | (64, 64, 64) | 36,928 |
| Batch Normalization | (64, 64, 64) | 256 |
| Activation (ReLU) | (64, 64, 64) | 0 |
| Flattening | 262,144 | 0 |

[8] machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
[9] machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

| | | |
|---|---|---|
| Dense (512 nodes, ReLU activation) | 512 | 134,218,240 |
| Dense (256 nodes, ReLU activation) | 256 | 131,328 |
| Dense (124 nodes, ReLU activation) | 124 | 32,896 |
| Dense (2 nodes, Softmax activation) | 2 | 258 |

The baseline model trained on data with no image augmentation achieved a validation accuracy of 0.94650. When the pipeline was updated to include image augmentation, validation accuracy increased to 0.95650. This was surprisingly good performance for such a simple model, suggesting that there are clear differences between the damage and no damage classes that are relatively easy for the network to learn. However, there is still substantial room for improvement. Based on the performance increase associated with including image augmentation, pipelines for all models below include image augmentation.

## Model Improvement and Refinement

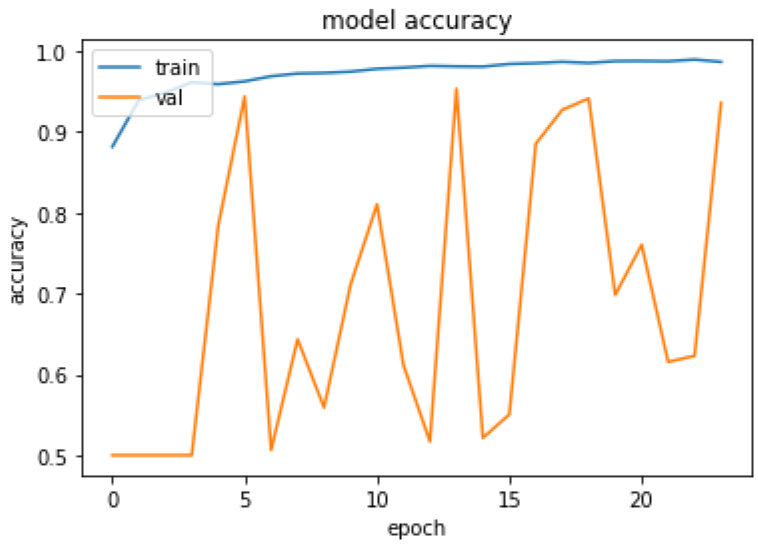
During training of the baseline model, training validation scores regularly exceeded 0.99, while the validation accuracy was substantially lower. This suggested that the model was overfitting, even with variation in training data introduced by the image augmentation. Two common ways to reduce overfitting for convolutional neural networks are to 1) add max pooling layers after each convolution layer and 2) add dropout layers after each dense layer. As an added benefit, both these techniques also increase training efficiency.

Max pooling layers downsample the outputs of the convolution layers by sliding a filter of the outputs and calculating the maximum pixel value within each window. This reduces the sensitivity of the model to exactly where features are located in an input image. Max pooling layers were added before the activation functions as this reduces the number of features that the activation function has to work on, increasing computational efficiency.

Dropout layers reduce overfitting and improve generalization by randomly subsampling the nodes in the dense layers (“dropping out” others). Dropout layers have the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs[10] - reducing the chances that the model will overfit to noise in the training data. Because dropout layers thin the network during training, it is sometimes necessary to increase the number of nodes in the dense layers to compensate.

Several combinations of hyperparameters were tested including smaller and larger convolution filters and more or less nodes in the dense layers. While most of the models performed relatively well, a recurring issue was that models would achieve high accuracy after only a few training epochs but never converge. The figure below summarizes training and validation accuracy by epoch for a model that demonstrates this trend:

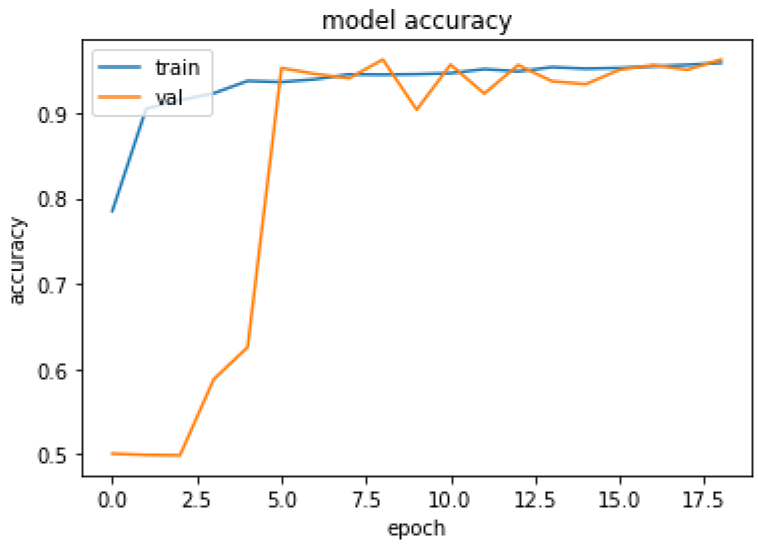
[10] https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/



This lack of convergence suggests that while the model did achieve high accuracy on the validation set, it may not be generalizable to another set of unseen data (the test set). Model convergence was ultimately improved by a combination of several updates to the model architecture:

- Reduction in kernel size and stride for the first convolution layer
- Reduction in number of filters in first convolution layer
- Reduction in number of nodes in each dense layer by 50%
- Reduction in initial learning rate for the Adam optimizer from the default (0.001) to 0.0001

As summarized in the figure below, the updated model converges much better than before:



| Layer | Output Shape | # of Params |
|---|---|---|
| Rescaling | (128, 128, 3) | 0 |
| Convolution (filters=32, kernel_size=3, strides=1) | (128, 128, 32) | 896 |
| Max Pooling (pool size=2, strides=2) | (64, 64, 32) | 0 |
| Batch Normalization | (64, 64, 32) | 128 |
| Activation (ReLU) | (64, 64, 32) | 0 |
| Convolution (filters=32, kernel_size=3, strides=1) | (32, 32, 64) | 18,496 |
| Max Pooling (pool size=2, strides=2) | (16, 16, 64) | 0 |
| Batch Normalization | (16, 16, 64) | 256 |
| Activation (ReLU) | (16, 16, 64) | 0 |
| Convolution (filters=32, kernel_size=3, strides=1) | (8, 8, 64) | 36,928 |
| Max Pooling (pool size=2, strides=2) | (4, 4, 64) | 0 |
| Batch Normalization | (4, 4, 64) | 256 |
| Activation (ReLU) | (4, 4, 64) | 0 |
| Flattening | 1024 | 0 |
| Dense (512 nodes, ReLU activation) | 512 | 524,800 |
| Dropout (rate=0.3) | 512 | 0 |
| Dense (256 nodes, ReLU activation) | 256 | 131,328 |
| Dropout (rate=0.2) | 256 | 0 |
| Dense (128 nodes, ReLU activation) | 128 | 32,896 |
| Dropout (rate=0.1) | 128 | 0 |
| Dense (2 nodes, Softmax activation) | 2 | 258 |

This updated model achieved a validation accuracy of 0.9735, a substantial improvement over the baseline model.

## Transfer Learning Leveraging Deep Learning Models

After training and evaluating the models described above, I next leveraged transfer learning to find out if pre-trained deep learning models could produce better and/or more stable results. Out of the many deep learning models available I decided to use Resnet50[11] because a) it offers a

[11] https://arxiv.org/abs/1512.03385

good balance of accuracy and training efficiency and b) it was used in the baseline model for the xView2 competition[12], which suggested that it would perform well for a similar aerial imagery classification task.

The Resnet50 model, along with weights trained on ImageNet, was inserted between the convolution and dense layers of the best performing model architecture from the prior experiments. Resnet model weights were frozen, with the additional convolution and dense layers trained on the structure damage training dataset.

Starting with the same hyperparameter settings for the convolution and dense layers as the best performing model from the previous section, several iterations based on different combinations of convolution filters size, dense layer node density, dropout rate, and learning rate were tested. The architecture of the best performing transforming model is summarized below (with the deep Resnet layers as a single line for brevity).

| **Layer** | **Output Shape** | **# of Params** |
|---|---|---:|
| Rescaling | (128, 128, 3) | 0 |
| Convolution (filters=32, kernel_size=5, strides=2) | (64, 64, 32) | 2,432 |
| Max Pooling (pool size=2, strides=2) | (32, 32, 32) | 0 |
| Batch Normalization | (32, 32, 32) | 128 |
| Activation (ReLU) | (32, 32, 32) | 0 |
| Convolution (filters=32, kernel_size=3, strides=1) | (16, 16, 64) | 18,496 |
| Max Pooling (pool size=2, strides=2) | (8, 8, 64) | 0 |
| Batch Normalization | (8, 8, 64) | 256 |
| Activation (ReLU) | (8, 8, 64) | 0 |
| Convolution (filters=32, kernel_size=3, strides=1) | (4, 4, 64) | 36,928 |
| Max Pooling (pool size=2, strides=2) | (2, 2, 64) | 0 |
| Batch Normalization | (2, 2, 64) | 256 |
| Activation (ReLU) | (2, 2, 64) | 0 |
| Flattening | 256 | 0 |
| FROZEN RESNET LAYERS | | |
| Dense (1024 nodes, ReLU activation) | 1024 | 263,168 |
| Dropout (rate=0.3) | 1024 | 0 |
| Dense (512 nodes, ReLU activation) | 512 | 524,800 |

[12] https://github.com/DIUx-xView/xView2_baseline

|  |  |  |
|---|---|---|
| Dropout (rate=0.2) | 512 | 0 |
| Dense (256 nodes, ReLU activation) | 256 | 131,328 |
| Dropout (rate=0.1) | 256 | 0 |
| Dense (2 nodes, Softmax activation) | 2 | 258 |

## Comparing Models

The table below summarizes a select subset of the iterations evaluated (not all are included for brevity). Note that all models except the initial baseline model included image augmentation in their pipeline (rotate, flip, and zoom).

| **Model** | **Validation Accuracy** |
|---|---|
| Baseline (no image augmentation) | 0.9365 |
| Baseline | 0.9440 |
| With Max Pooling (kernel=5) & Dropout Layers | 0.9500 |
| With Max Pooling (kernel=10) & Dropout Layers | 0.9230 |
| **With Max Pooling (kernel=3) & Dropout Layers (dense layers with 50% less nodes)** | **0.9735** |
| Transfer Learning (with Max Pooling kernel=5) | 0.9765 |
| Transfer Learning (Max Pooling kernel=3) | 0.9735 |

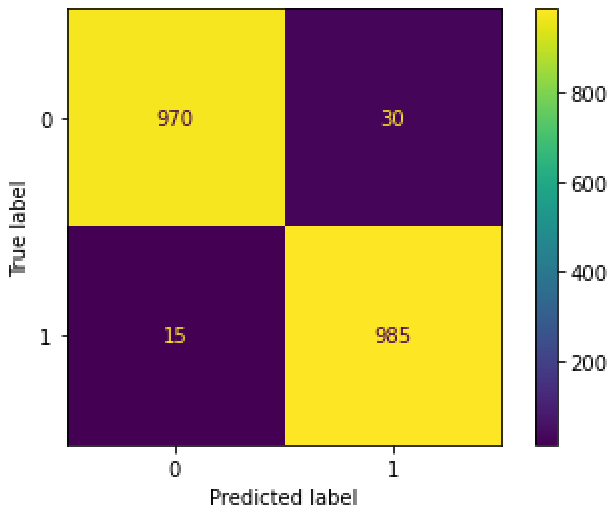
An unexpected result was that while the transfer learning model did perform slightly better than the best performing standard model, the difference in validation accuracy was only 0.003 - which corresponds to 6 images in the validation set. This is well within the potential variation that would be expected if a different set of images had been randomly selected for the validation set. Therefore it cannot be said with certainty that the transfer learning model performs better than the standard model (cross validation was not employed due to time limitations).

During testing it was noted that the standard model is significantly smaller in size (11 mb vs. 500 mb) and prediction is much more time and computationally efficient. Because the validation accuracy rates were essentially the same, and the standard model was more computational efficient, it was selected as the final model.

## Final Model Performance[13]

The final model was then evaluated based on test data (an additional 2,000 images which had been kept separate from the training and validation sets). The final model achieved a **test accuracy of 0.9775**, which was even higher than the validation accuracy (0.9735). Similar performance on the validation and test sets suggest that the model would be generalizable to additional unseen data (see caveats in Conclusion below).

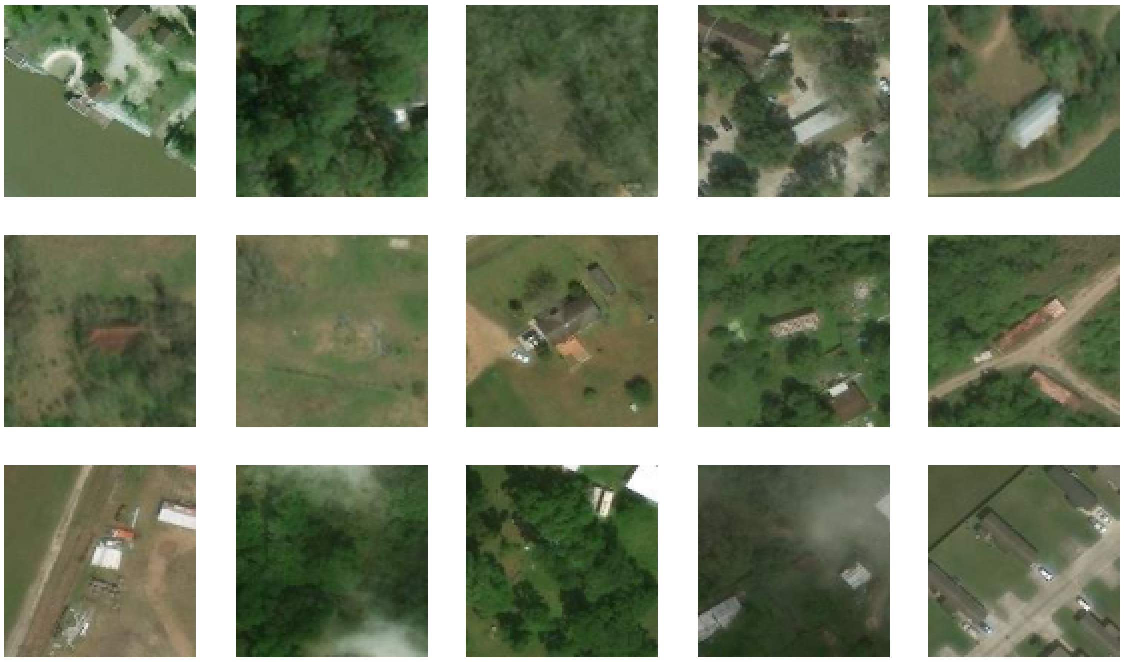
A confusion matrix of the models predictions on test data reveal that the false positives were twice as common as false negatives (0=no damage, 1=damage):



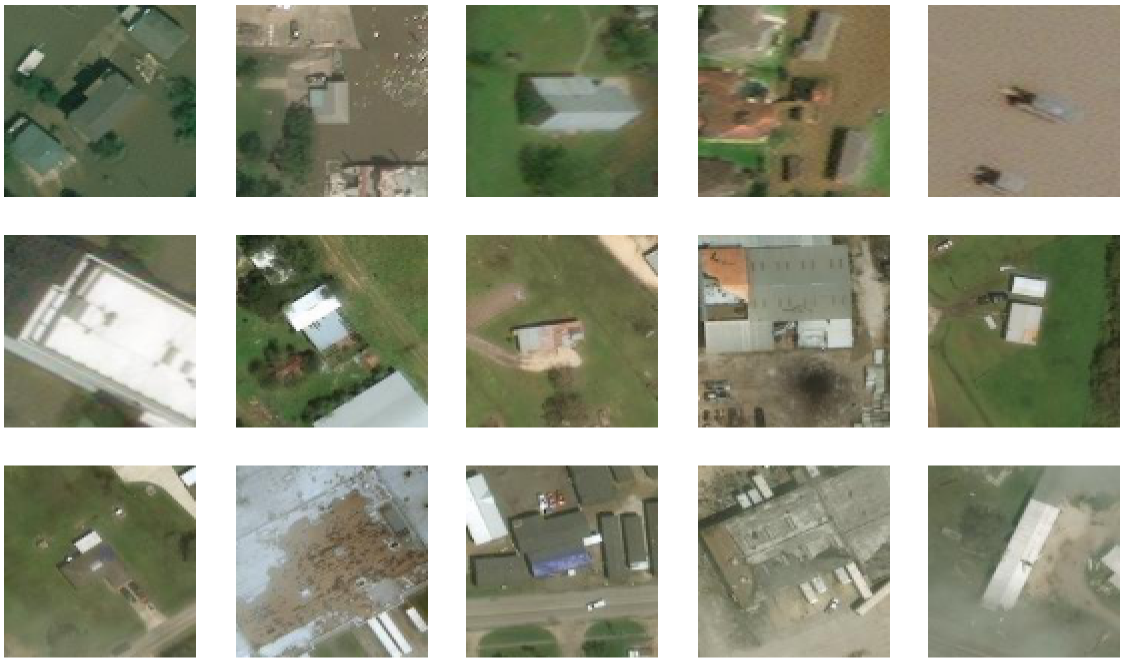
Examining the misclassified images reveals some insights about the model:

[13] Notebook: https://github.com/allankapoor/wildfire_prediction/blob/master/Step4_FinalModelEval.ipynb

**False Positives** (30 total, 15 shown) tend to have surfaces that are mistaken for flood waters or junk around them that is mistaken for damage. False positives also tend to be rural structures:



**False negatives** (15 total) appear to be mostly large or non-residential structures, have a lot of variation in the ground surface, and/or no obvious flood water:



# Conclusion

## Caveats and Potential Improvements

This effort produced a model that performs well on both validation and test data. However, there are some important caveats and limitations to be aware of, which could be addressed through access to improved (or different) training data.

Many (but not all) of the images depicting damaged structures included visible flood water, suggesting that the model likely learned to identify flood water (among other features). A model that could identify structure damage even after flood waters have subsided would be even more useful but would need to identify much more subtle attributes in the images. This could be accomplished with labeled training images from imagery several weeks after a hurricane, rather than directly after.

From plotting the geographic location of each of the images in the test dataset it is clear that while the images all come from the Houston area, in many cases the damaged and undamaged structures are from different neighborhoods. This is to be expected given that flooding is influenced by topography and proximity to rivers, but this also means that it is possible that there tended to be differences between the damaged and undamaged structures beyond differences caused by the flooding. For example, different neighborhoods may have differently sized structures, various styles of architecture, or differing levels of tree canopy. In addition to damage/flood features, the model may have also learned to identify these other differences that are irrelevant to the problem statement. This could be addressed by limiting the training data to areas where both damaged and not damaged. These areas could be identified with basic geospatial processing, but would require additional training data.

A similar potential issue is that because all the training images come from the same city, the model is likely not generalizable to make accurate classifications on images of structures from cities that have a different urban form or natural environment than Houston (for example, a dense urban area like New York City). This could be addressed by training the model with post-Hurricane images from multiple cities.

## Using the Model

This effort resulted in a model which could classify with high accuracy between aerial images of damaged and undamaged structures. The model would be most useful as part of a pipeline that could directly ingest much larger aerial images, crop images of the individual structures, and then make the classification. As building detection algorithms are already quite accurate this could be accomplished relatively easily, either by leveraging an existing dataset of building footprints or by incorporating a building detection algorithm at the start of the pipeline. For example, Microsoft has produced building footprints for the entire US that are publicly available in GeoJSON format[14]. Leveraging this dataset, a pipeline could be developed which would

[14] https://github.com/microsoft/USBuildingFootprints

ingest a single large aerial image (or images), crop tiles based on the location of building footprints, classify the images as damaged/not damaged, and then plot these classifications as points on a map. This pipeline could effectively transform post-hurricane aerial imagery into a city-wide damage assessment map with little human effort beyond the initial flight. Use of satellite imagery would reduce the level of effort even further. This would be a valuable tool for federal to local first responders.